BAB 5 PERANCANGAN DAN IMPLEMENTASI

5.1 Perancangan Sistem

Perancangan sistem digunakan untuk mempermudah memahami kebutuhan dari sistem. Dalam perancangan sistem terdapat tiga diagram yaitu diagram *sequence*, *class diagram* dan *entity relationship diagram*. Diagram *sequence* untuk menjelaskan aliran cara kerja sistem dari diagram *use case* berdesarkan waktu. Serta pada perancangan sistem juga menggunakan c*lass diagram* untuk menggambarkan kelas-kelas yang dibutuhkan dalam sistem dan *Entity Relationship Diagram* digunakan untuk menggambarkan entitas yang diperlukan untuk perancangan basis data. Pada perancangan sistem juga dibuat perancangan antarmuka sebagai gambaran antarmuka sistem yang akan diimplementasikan.

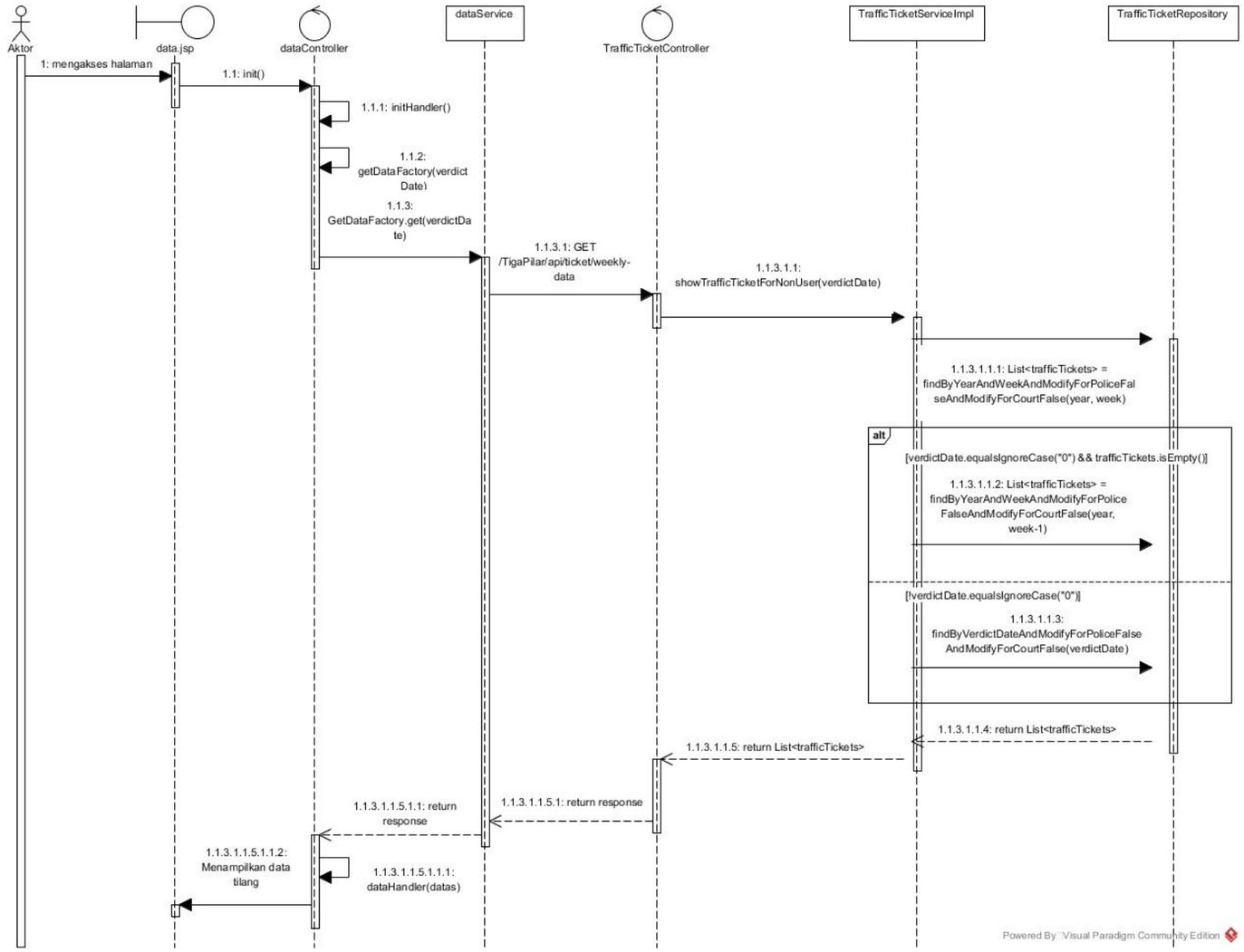
5.1.1 Diagram *sequence*

Diagram *sequence* digunakan untuk menggambarkan alur atau jalannya proses dalam sistem berdasarkan urutan waktu. Pada sub-bab ini akan digambarkan diagram *sequence* dari sistem. Diagram *sequence* yang digambarkan pada sub-bab ini tidak semua dan hanya diambil yang penting saja dari sistem, untuk melihat diagram *sequence* yang lain dapat dilihat pada lampiran B.

- **Diagram sequence iterasi pertama**

Diagram *sequence* iterasi pertama dibuat berdasarkan kebutuhan fungsional pada iterasi pertama. Diagram *sequence* iterasi pertama dapat dilihat pada gambar 5.1 sampai dengan gambar 5.14.

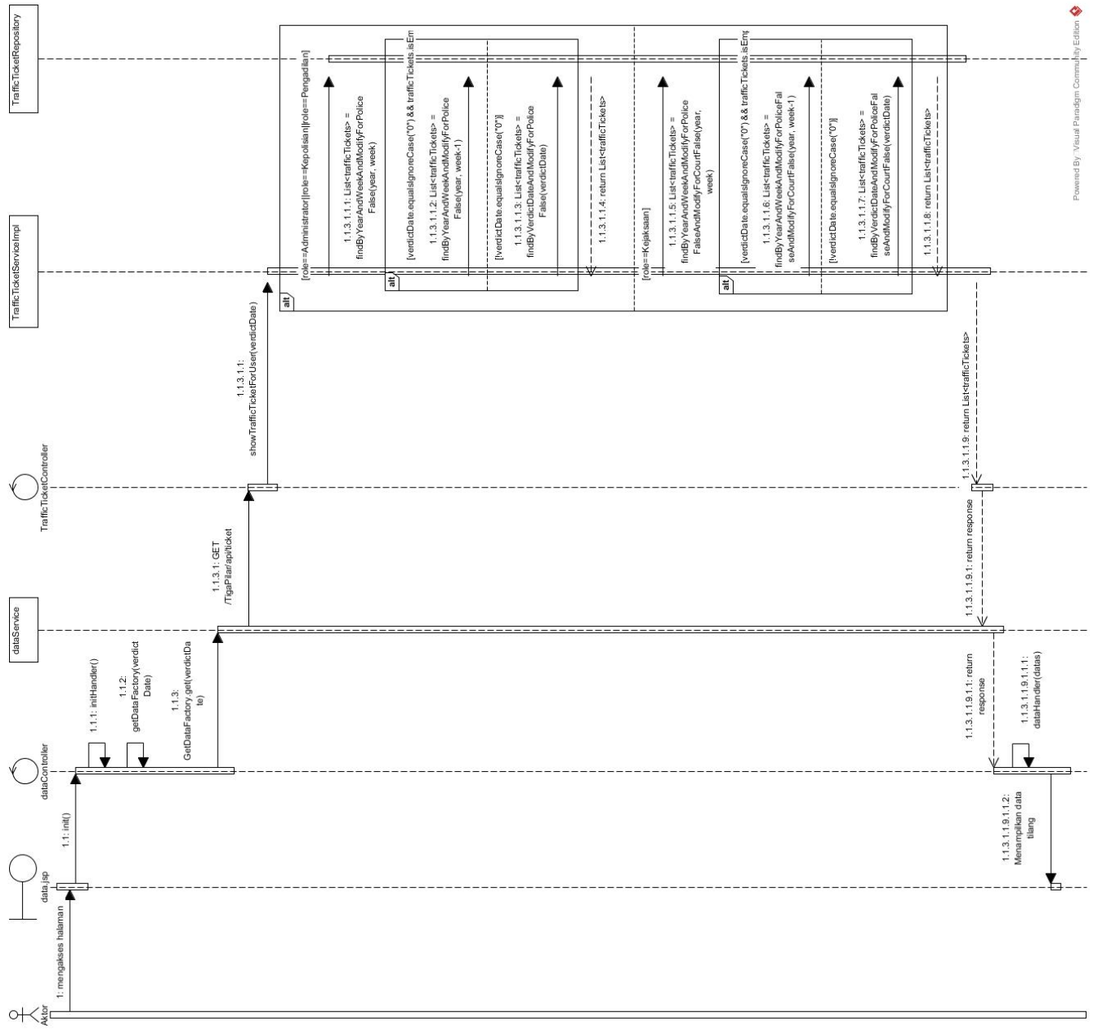
5.1.1.1 Diagram *sequence* melihat data tilang *non user*



Gambar 5.1 Diagram *sequence* melihat data tilang *non user*

Diagram *sequence* melihat data tilang *non user* dapat dilihat pada gambar 5.1. alur dari diagram *sequence* ini adalah aktor mengakses halaman awal *website* tiga pilar. Kemudian pada AngularJs akan memanggil *method* `init()` pada dataController dimana *method* tersebut akan dipanggil pertama kali ketika halaman diakses, selanjutnya dataController akan memanggil *method* `GetDataFactory.get(verdictDate)` yang akan berkomunikasi dengan *web service*. Pada *web service request* yang dilakukan oleh AngularJs akan diterima oleh TrafficTicketController yang akan memanggil *method* `showTrafficTicketForNonUser(verdictDate)` TrafficTicketServiceImpl. Pada *service* inilah *business logic* utama dari sistem, pada *service* ini akan dicek apakah *request* tanggal putusan adalah 0 atau tidak. Jika tanggal putusan bernilai 0 dan data minggu ini masih kosong, maka akan meanggil *method* `findByYearAndWeekAndModifyForPoliceFalseAndModifyForCourtFalse(year, week)` TrafficTicketRepository untuk menampilkan dat pada minggu sebelumnya, jika nilai tanggal putusan tidak 0, maka akan ditampilkan data tilang berdasarkan tanggal putusan dengan memanggil *method* `findByVerdictDateAndModifyForPoliceFalseAndModifyForCourtFalse(verdictDatte)` pada TrafficTicketRepository.

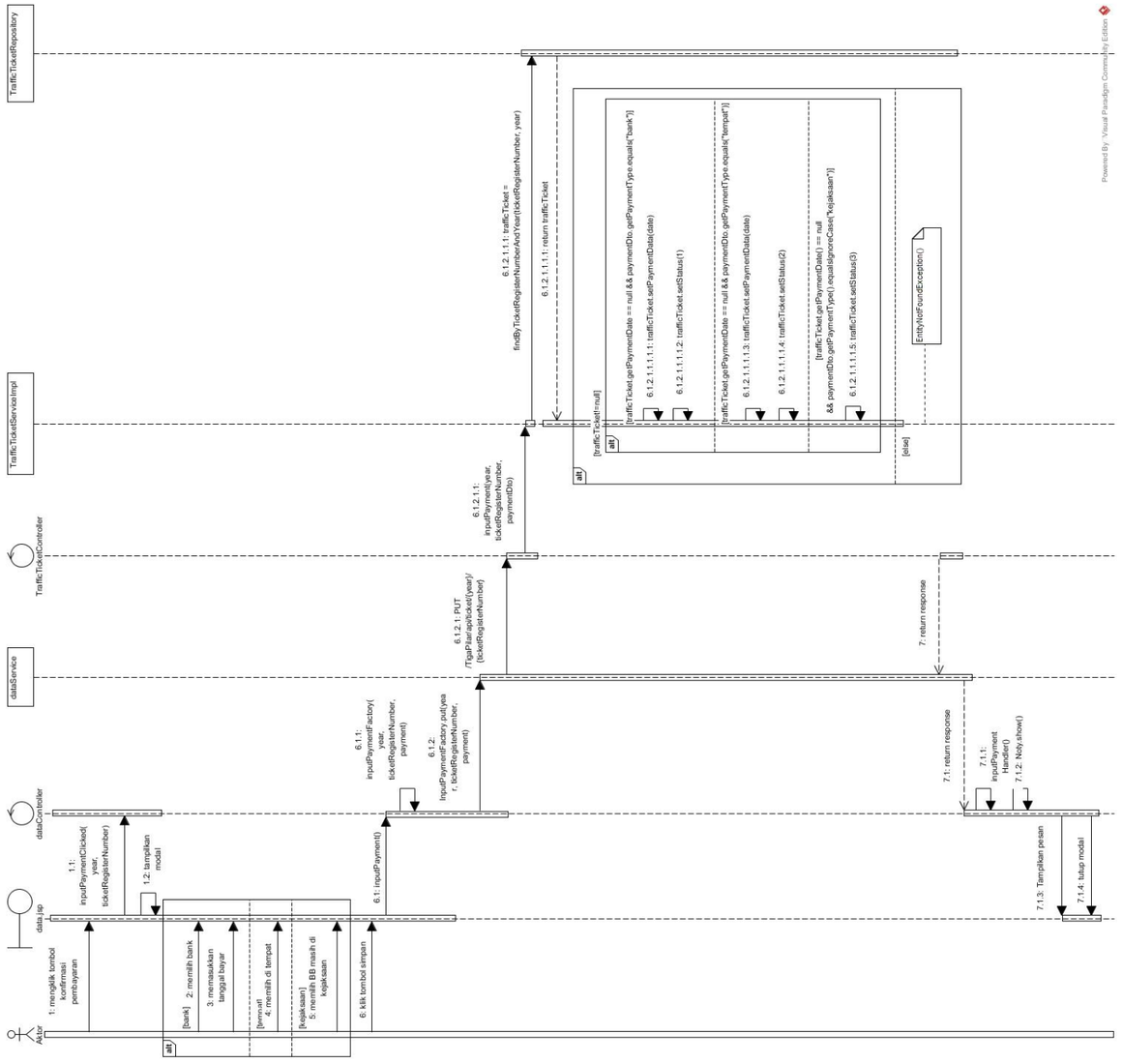
5.1.1.2 Diagram *sequence* melihat data tilang *user*



Gambar 5.2 Diagram *sequence* melihat data tilang *user*

Diagram *sequence* melihat data tilang *user* dapat dilihat pada gambar 5.2. Pada diagram ini digunakan untuk menampilkan data tilang bagi aktor yang mempunyai hak akses. Alur dari diagram *sequence* ini, aktor mengakses halaman data tilang, kemudian pada AngularJs akan memanggi *method* `init()` pada dataController, kemudian memanggil `GetDataFactory.get(verdictDate)` pada dataService yang akan melakukan *request* ke *web service*. *Request* yang dilakukan tadi akan diterima oleh TrafficTicketController, yang akan memanggil *method* `showTrafficTicketForUser(verdictDate)` pada TrafficTicketServiceImpl. Pada *service* inilah *business logic* dari sistem, jika *user* yang mengakses memiliki *role* administrator, atau kepolisian, atau pengadilan maka akan memanggil *method* `findByYearAndWeekAndModifyForPoliceFalse(year, week)` pada TrafficTicketRepository, sedangkan jika *user* memiliki *role* kejaksaan maka akan memanggil *method* `findByYearAndWeekAndModifyForPoliceFalseAndModifyForCourtFalse(year, week)` hal ini dikarenakan akses data untuk kejaksaan baru boleh melihat data tilang ketika telah diputus oleh pengadilan.

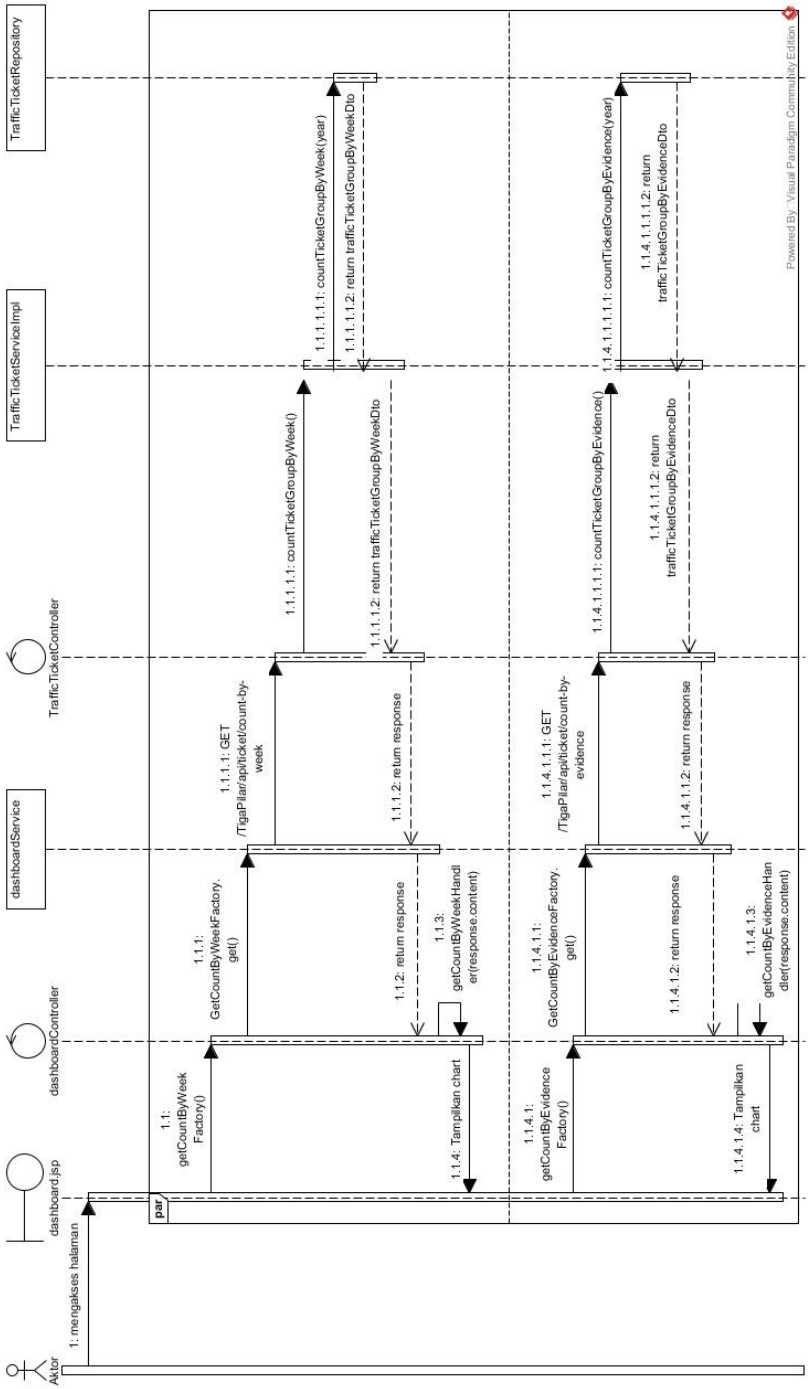
5.1.1.3 Diagram *sequence* konfirmasi pembayaran tilang



Gambar 5.3 Diagram *sequence* konfirmasi pembayaran tilang

Diagram *sequence* konfirmasi pembayaran dapat dilihat pada gambar 5.3. pada diagram ini dijelaskan alur untuk konfirmasi pembayaran tilang. Aktor sebagai operator kejaksaan akan mengakses halaman data tilang, kemudian menekan tombol konfirmasi pembayaran, dan memilih jenis pembayaran yang dilakukan seperti pembayaran di bank, pembayaran di tempat, dan barang bukti masih di kejaksaan, kemudian aktor akan menekan tombol simpan. Pada AngularJs akan memanggil *method* `inputPayment()` pada dataController yang akan memanggil *method* `inputPaymentFactory.put(year, ticketRegisterNumber, payment)` pada dataService. Kemudian *request* dari dataService akan diterima oleh TrafficTicketController yang akan memanggil *method* `inputPayment(year, ticketRegisterNumber, paymentDto)` pada TrafficTicketService. Disini akan dicek jenis pembayarannya jika pembayaran di bank maka status akan diset dengan nilai 1, jika pembayaran di tempat maka status akan diset dengan nilai 2, jika pembayaran di kejaksaan maka akan diset dengan nilai 3.

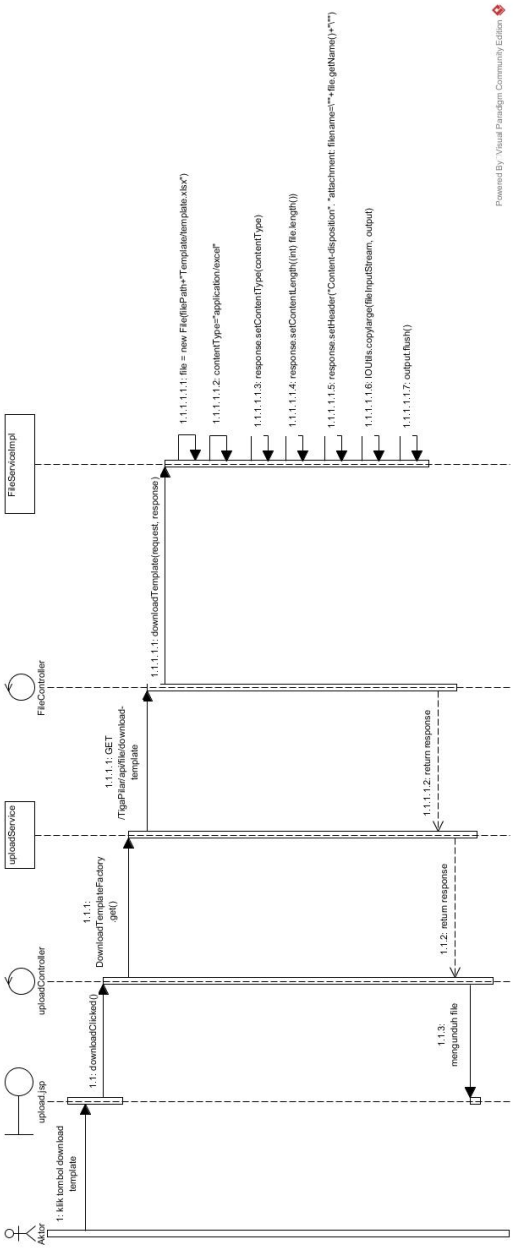
5.1.1.4 Diagram *sequence* melihat statistik data tilang per tahun



Gambar 5.4 Diagram *sequence* melihat statistik data tilang per tahun

Diagram *sequence* melihat statistik data tilang dapat dilihat pada gambar 5.4. pada diagram *sequence* ini akan memanggil dua *web service* sekaligus yaitu menampilkan total data tilang per minggunya dalam satu tahun, dan menampilkan total data tilang berdasarkan barang bukti dalam satu tahun yang akan ditampilkan dalam bentuk *chart*.

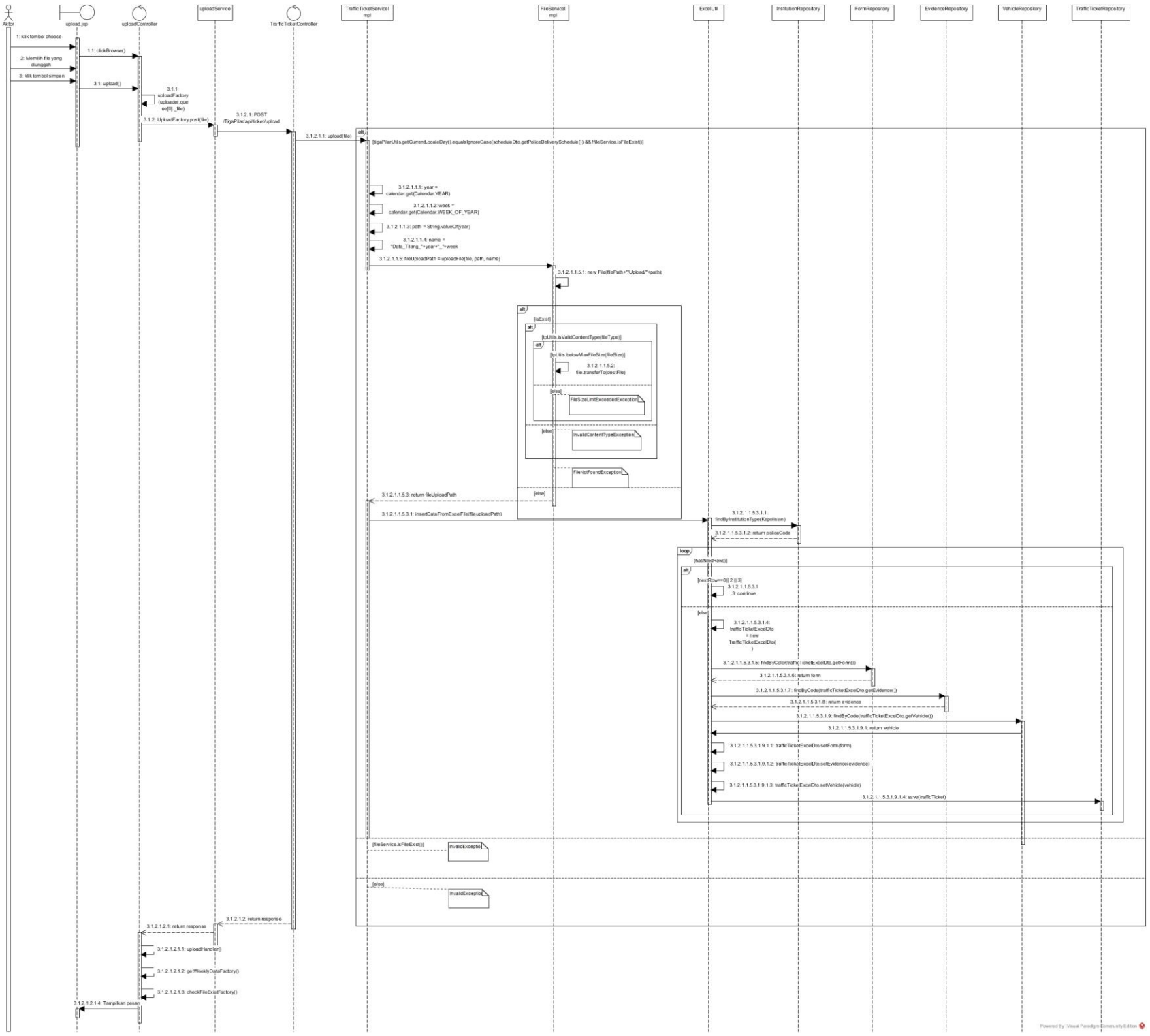
5.1.1.5 Diagram *sequence* mengunduh *template file* data tilang



Gambar 5.5 Diagram *sequence* mengunduh *template file* data tilang

Diagram *sequence* mengunduh *template file* data tilang dapat dilhat pada gambar 5.5. Pada diagram *sequence* ini menjelaskan alur bagaimana mengunduh *template file* data tilang. Aktor akan mengakses ke halaman *upload*, kemudian aktor akan menekan tombol *download template* yang akan memanggil *method* `downloadClicked()` pada uploadController yang akan memanggil *method* `DownloadTemplateFactory.get()` pada uploadService di AngularJs. Kemudian *request* dari AngularJs akan diterima oleh FileController pada *web service* yang akan memanggil *method* `dowloadTemplate(request, response)` pada FileServiceImpl. Pada *service* inilah *file* dengan *file path* /template/template.xlsx akan diunduh.

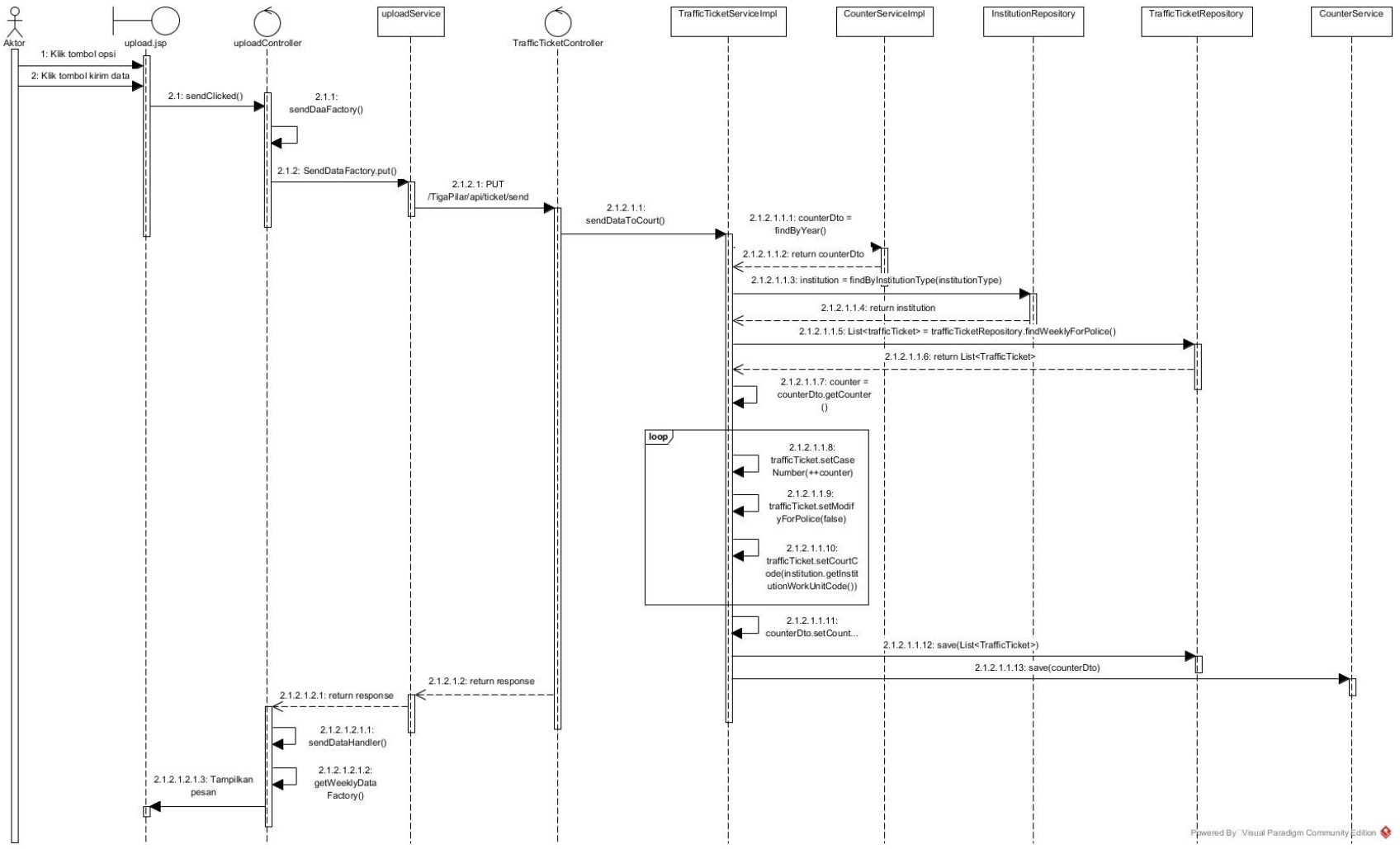
5.1.1.6 Diagram *sequence* mengunggah *file* data tilang



Gambar 5.6 Diagram *sequence* mengunggah *file* data tilang

Diagram *sequence* mengunggah *file* data tilang dapat dilihat pada gambar 5.6. Alur secara umum dari diagram ini adalah melakukan pengecekan hari pengiriman data tilang, dan *file* data tilang telah diunggah. Kemudian *file* tersebut akan disimpan pada *directory* tertentu, lalu setelah disimpan pada *directory* tertentu *file* tersebut akan dibaca isinya dan dimasukkan ke basis data per barisnya.

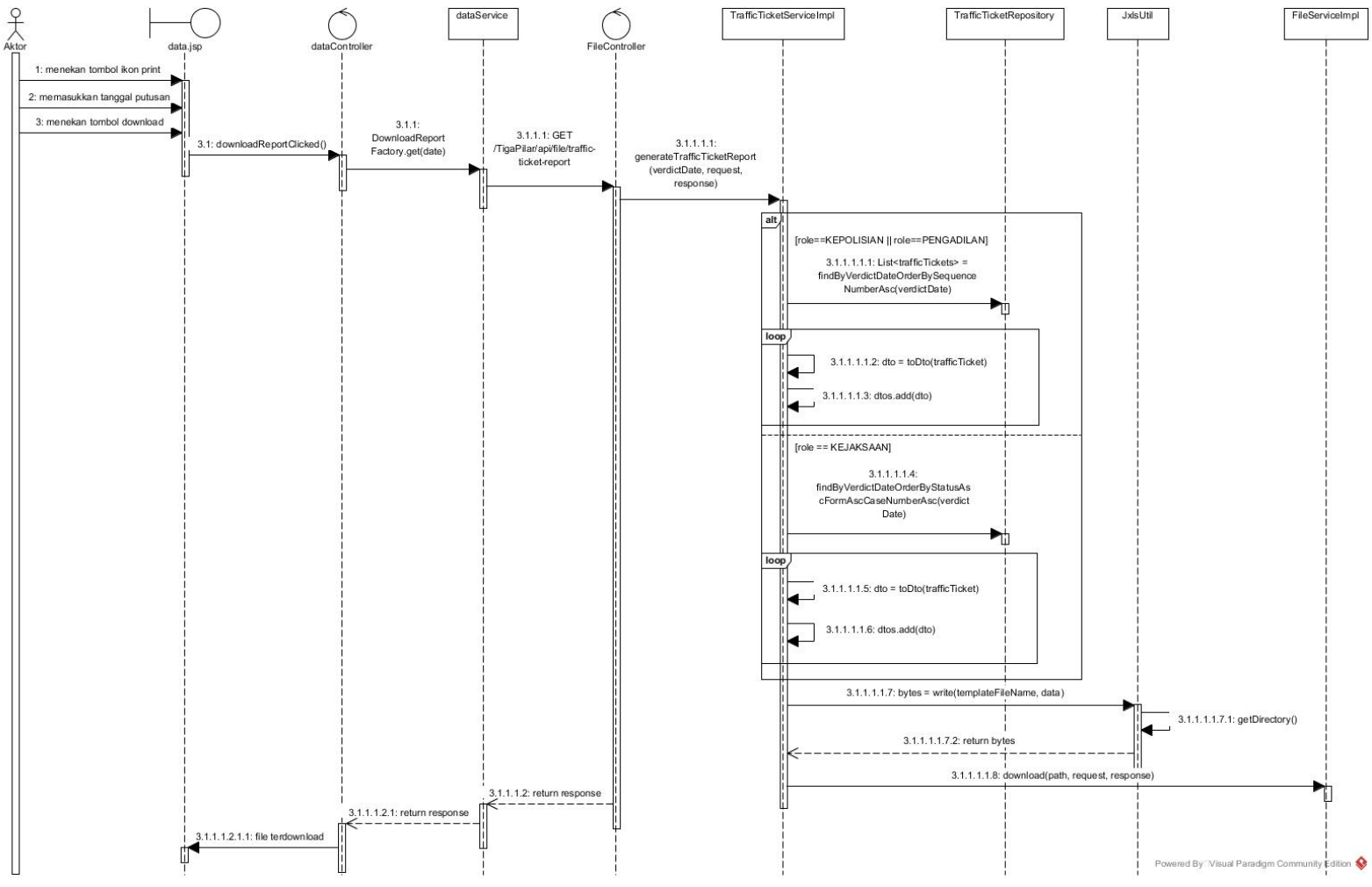
5.1.1.7 Diagram *sequence* mengirimkan data tilang



Gambar 5.7 Diagram *sequence* mengirimkan data tilang

Diagram *sequence* mengirimkan data tilang dapat dilihat pada gambar 5.7. Diagram *sequence* ini menjelaskan alur untuk mengirimkan data tilang yang telah diunggah oleh kepolisian ke pengadilan. Aktor menekan tombol kirim data yang akan memanggil *method* `sendClicked()` pada uploadController, yang selanjutnya dari *controller* ini akan memanggil *method* `sendDataFactory.put()` pada uploadService di AngularJs. *Request* dari AngularJs akan diterima oleh *web service* yang akan memanggil *method* `sendDataToCourt()` pada TrafficTicketServiceImpl. Pada *service* ini akan diset nomor perkara dan kode satuan kerja pengadilan.

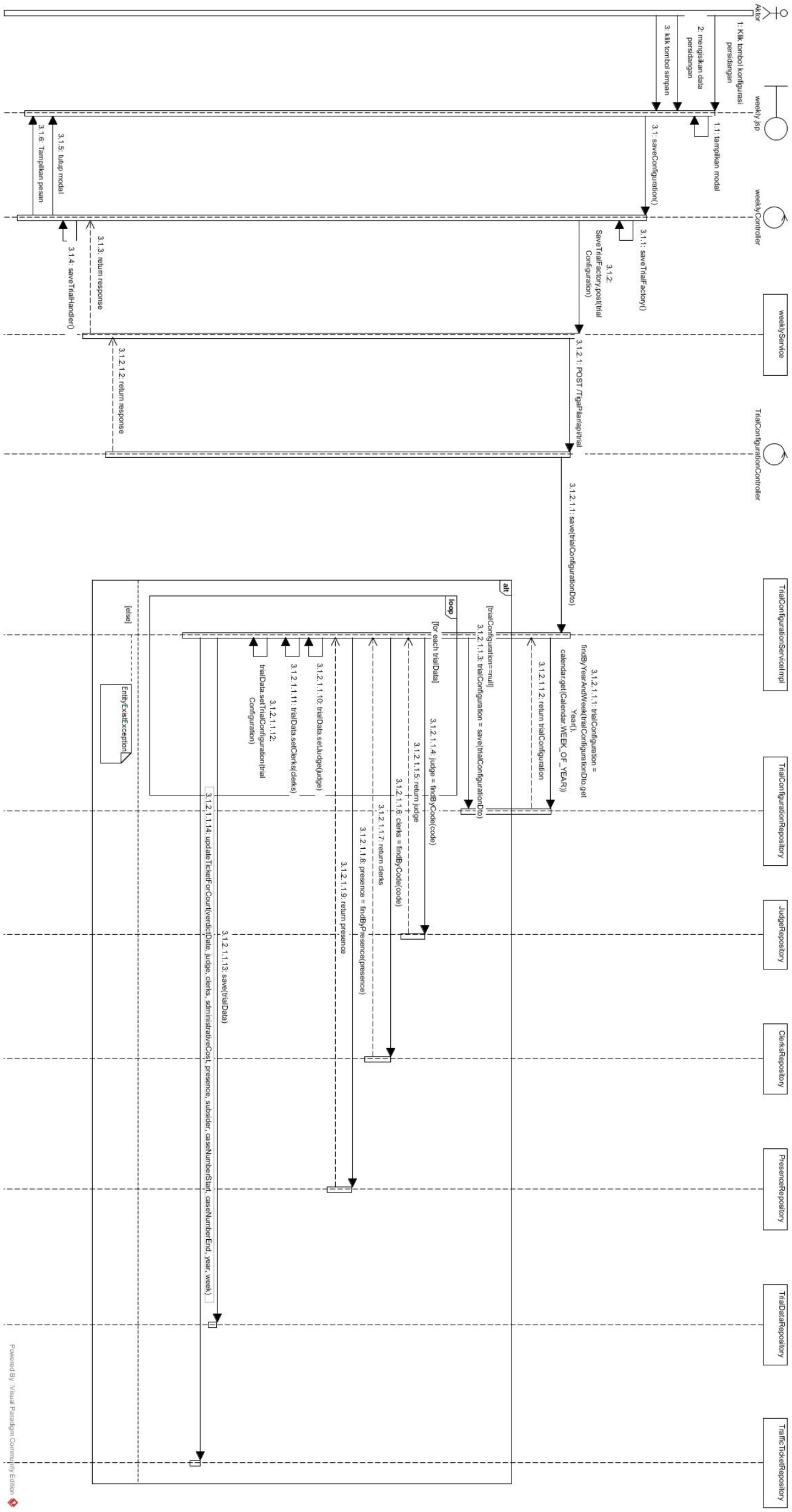
5.1.1.8 Diagram *sequence* mengunduh laporan data tilang mingguan



Gambar 5.8 Diagram *sequence* mengunduh laporan data tilang mingguan

Diagram *sequence* mengunduh laporan data tilang mingguan dapat dilihat pada gambar 5.8. Pada diagram *sequence* ini akan ada 2 alur yaitu jika *user* memiliki *role* kepolisian, atau pengadilan maka data tilang yang dibuat diurutkan berdasarkan nomor perkara, sedangkan jika *user* memiliki *role* kejaksaan maka data tilang yang dibuat diurutkan berdasarkan jenis *form* tilang dan status.

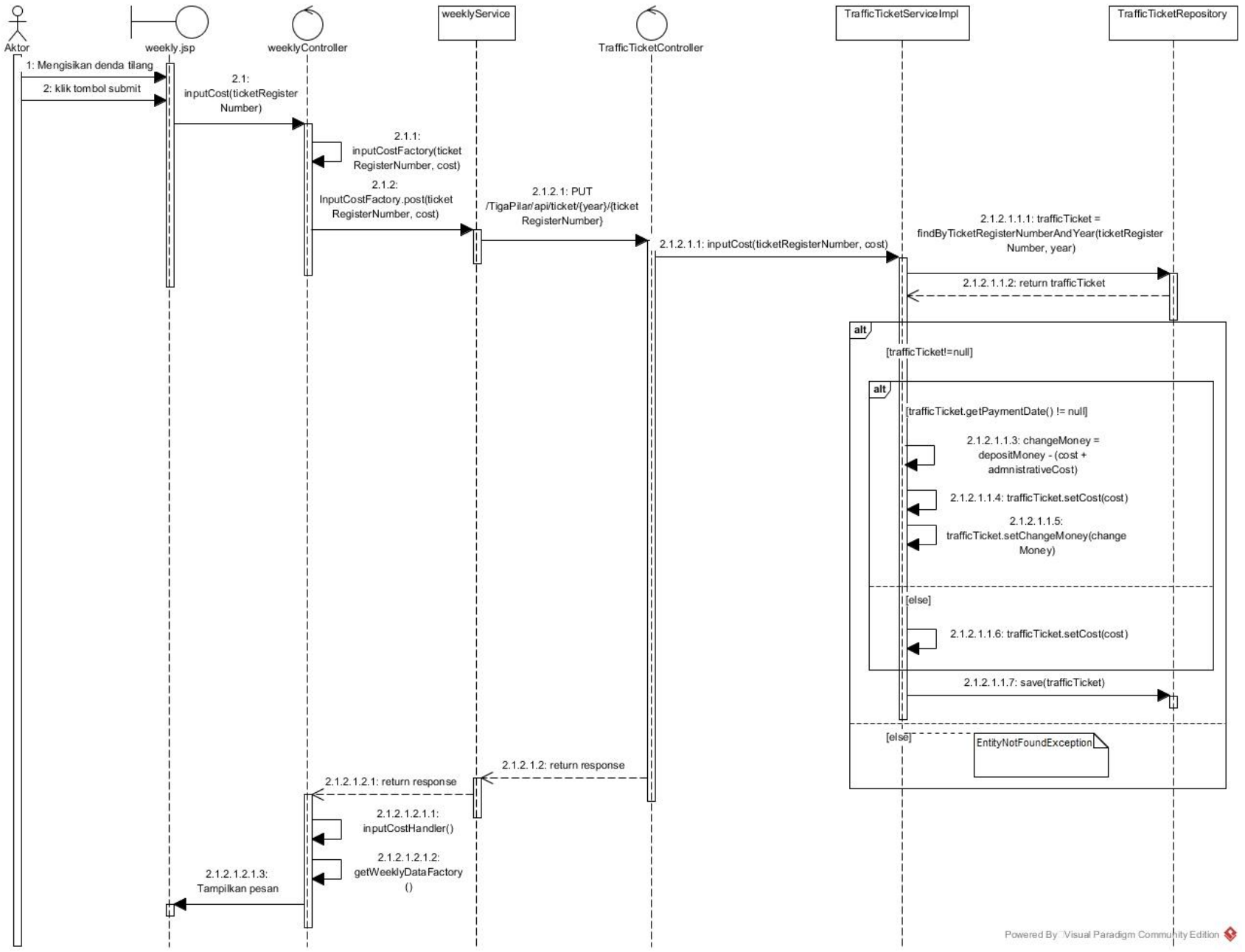
5.1.1.9 Diagram *sequence* memasukkan konfigurasi persidangan



Gambar 5.9 Diagram *sequence* memasukkan konfigurasi persidangan

Diagram *sequence* memasukkan konfigurasi persidangan dapat dilihat pada gambar 5.9. Pada diagram ini menjelaskan alur untuk menyimpan konfigurasi persidangan seperti siapa hakim, panitera, ruangan sidang, dan putusan denda yang diberikan.

5.1.1.10 Diagram *sequence* memasukkan denda tilang



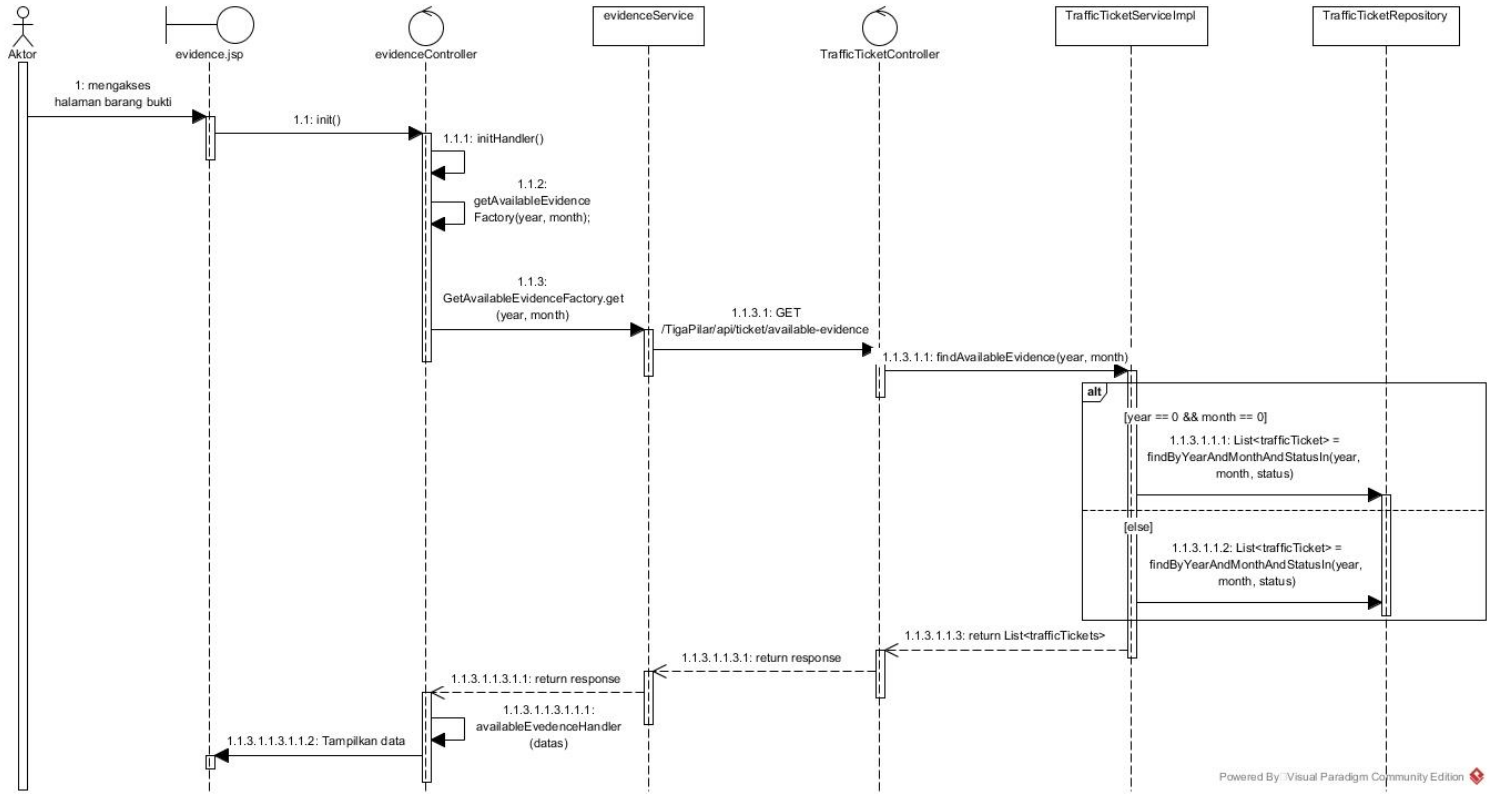
Gambar 5.10 Diagram *sequence* memasukkan denda tilang

Diagram *sequence* memasukkan denda tilang dapat dilihat pada gambar 5.10. Diagram *sequence* ini menjelaskan alur memasukkan denda tilang, dimana terdapat dua jalur. Jalur pertama ketika pada data tilang terdapat tanggal pembayaran maka akan dimasukkan besaran denda tilang yang diberikan, dan akan dikalkulasikan apakah terdapat kembalian atau tidak, jalur yang kedua jika tidak terdapat tanggal pembayaran maka hanya akan disimpan besaran denda tilangnya saja.

- **Diagram *sequence* iterasi kedua**

Setelah dilakukan iterasi pertama, ternyata terdapat penambahan kebutuhan fungsional dari *user.* Sehingga pada iterasi kedua juga terdapat penambahan diagram *sequence* untuk menjelaskan penambahan kebutuhan fungsional yang terjadi pada iterasi kedua.

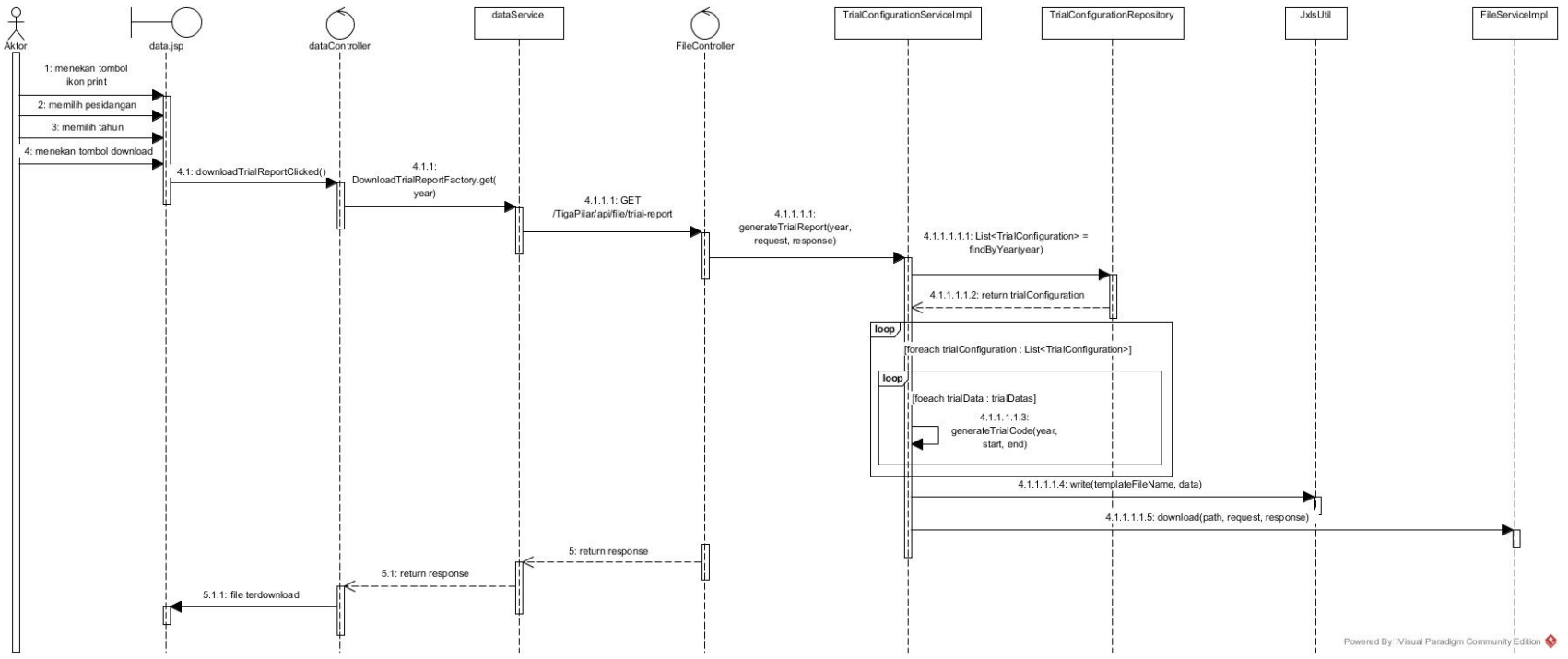
5.1.1.11 Diagram *sequence* menampilkan barang bukti yang belum diambil (iterasi kedua)



Gambar 5.11 Diagram *sequence* menampilkan barang bukti yang belum diambil (iterasi kedua)

Diagram *sequence* menampilkan barang bukti yang belum diambil dilakukan pada iterasi kedua dapat dilihat pada gambar 5.11. Aktor akan mengakses halaman barang bukti. Saat halaman ini diakses maka akan mengakses *method* `init()` pada evidenceController yang akan memanggil *method* `GetAvailableEvidence.get(year, month)` pada evidenceService di AngularJs. *Request* ini akan diterima oleh *web service* yang selanjutnya akan memanggil *method* `findAvailableEvidence(year, month)`. Jika nilai variabel *year* dan *month* bernilai 0 maka akan memanggil *method* `findByYearAndMonthAndStatusIn(year, month, status)` di TrafficTicketRepository pada bulan dan tahun saat ini. Jika nilai *year* dan *month* tidak 0 maka akan mencari data tilang dengan barang bukti yang belum diambil berdasarkan *input* dari *user*.

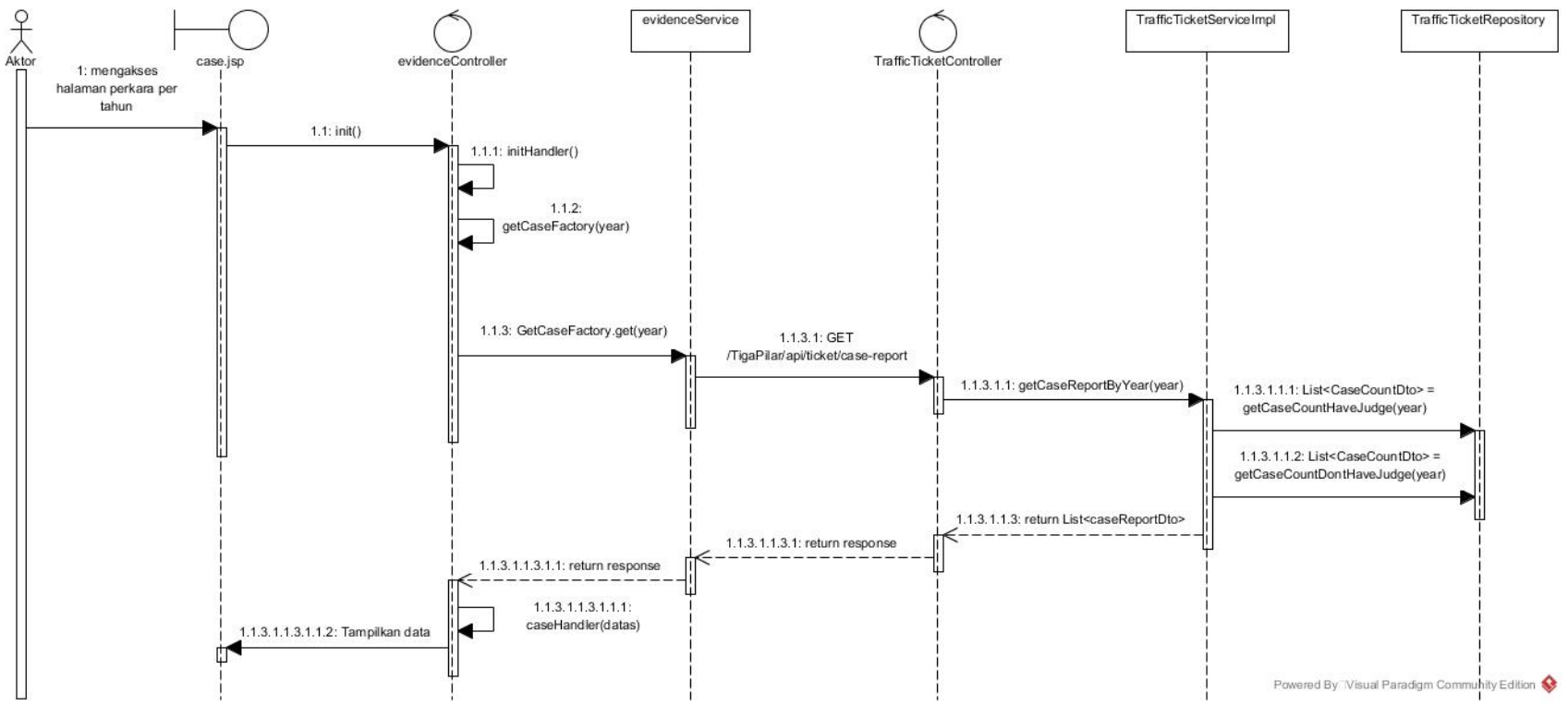
5.1.1.12 Diagram *sequence* mengunduh laporan persidangan (iterasi kedua)



Gambar 5.12 Diagram *sequence* mengunduh laporan persidangan (iterasi kedua)

Diagram *sequence* mengunduh laporan persidangan dilakukan pada iterasi kedua dapat dilihat pada gambar 5.12. Pada diagram *sequence* ini dijelaskan hal yang harus dilakukan aktor adalah menekan tombol dengan ikon *print*, kemudian memilih persidangan, memilih tahun, dan menekan tombol *download*. Pada sistem akan memanggil *method* `downloadReportClicked()` pada dataController di AngularJs, selanjutnya *controller* ini akan memanggil `DownloadReportFactory.get(year)` pada dataService di AngularJs. *Request* dari AngularJs akan diterima oleh *web service*, pada *web service* ini akan memanggil *method* `generateTrialReport(year, request, response)` pada TrialConfigurationServiceImpl untuk membuat laporan persidangan dalam satu tahun.

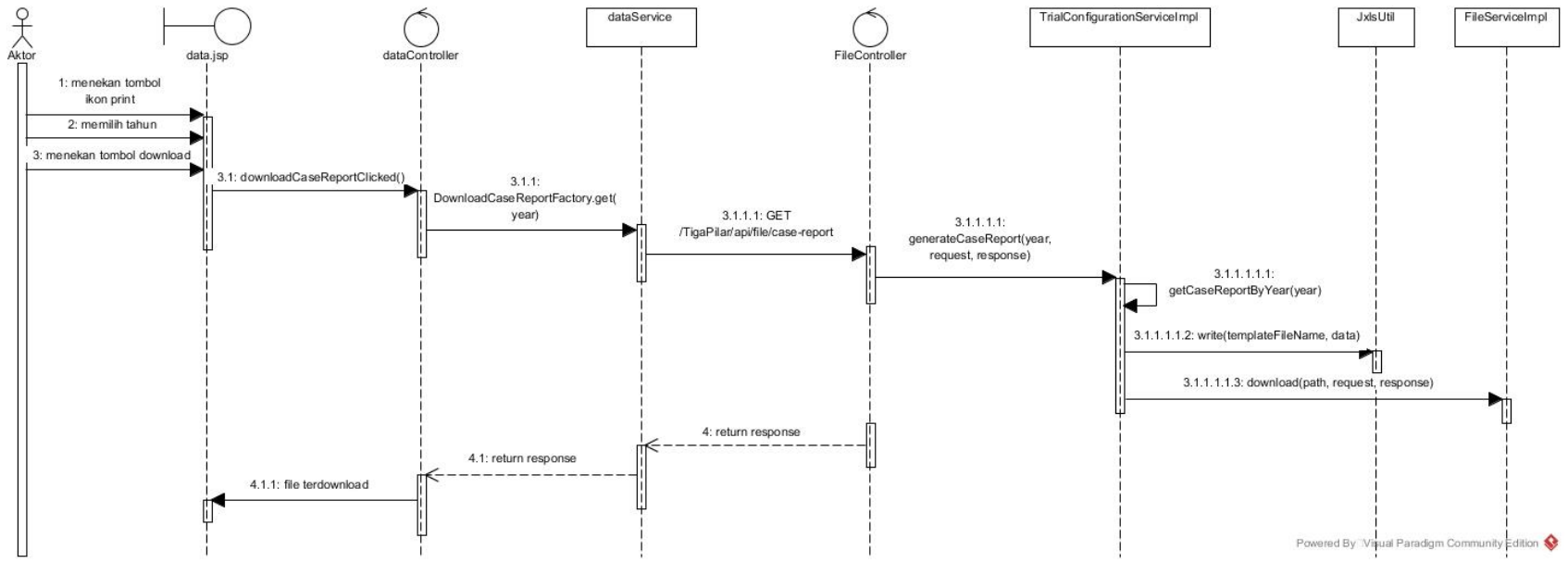
5.1.1.13 Diagram *sequence* menampilkan total perkara (iterasi kedua)



Gambar 5.13 Diagram *sequence* menampilkan total perkara (iterasi kedua)

Diagram *sequence* menampilkan total perkara dilakukan pada iterasi kedua dapat dilihat pada gambar 5.13. Pada diagram *sequence* ini dijelaskan hal yang harus dilakukan aktor adalah mengakses halaman perkara per tahun. Pada AngularJs caseController akan dipanggil *method* `init()` yang akan memanggil *method* `GetCaseReport.get(year)` pada caseService. Selanjutnya *request* tersebut akan diterima *web service* yang akan memanggil *method* `getCaseReportByYear(year)` pada TrafficTicketServiceImpl. Pada *method* tersebut akan mencari total data tilang yang sudah diputus oleh hakim dan data tilang yang belum diputus oleh hakim dengan memanggil *method* `getCaseCountHaveJudge(year)` dan *method* `getCaseCountDontHaveJudge(year)` pada TrafficTicketRepository.

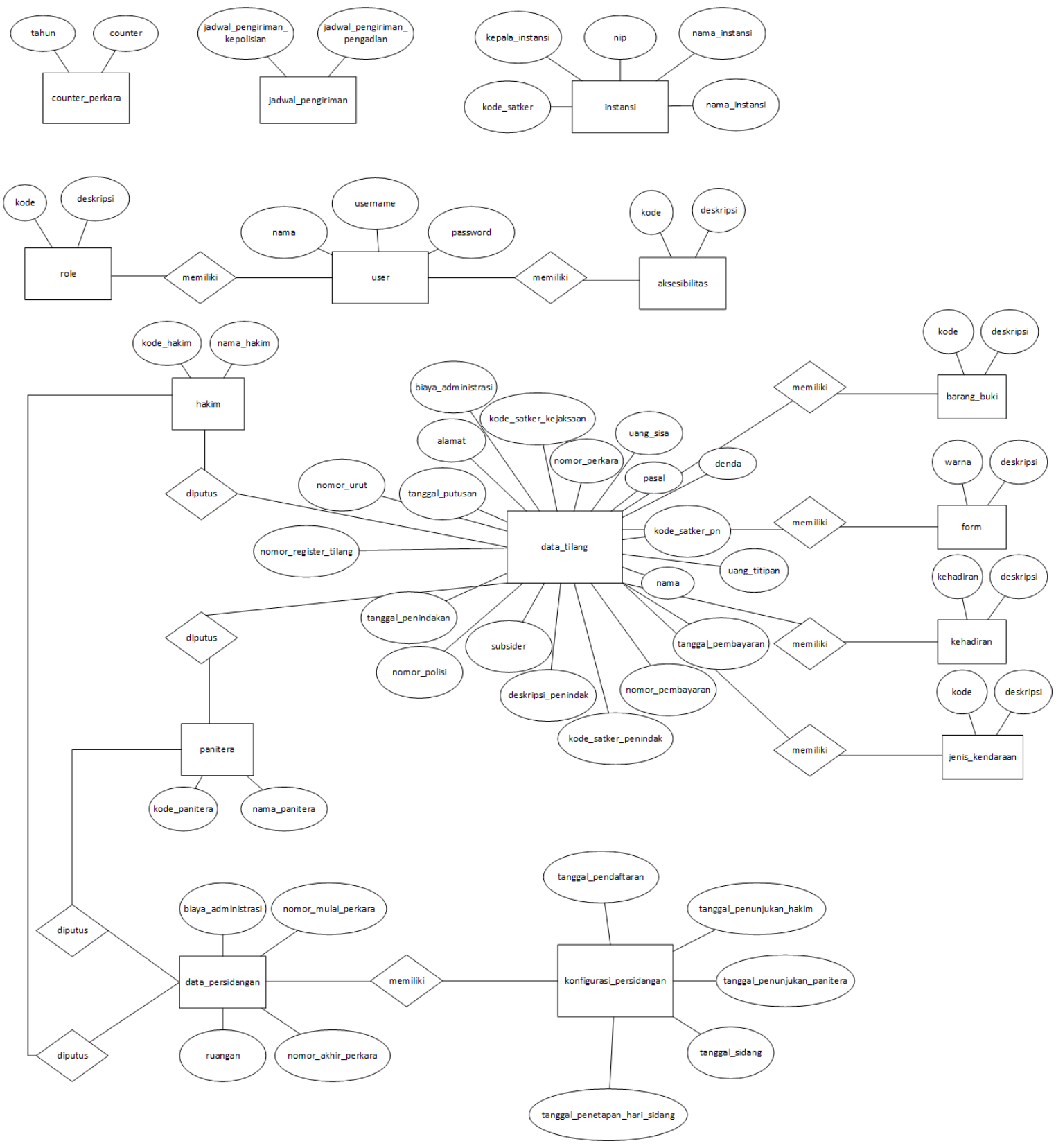
5.1.1.14 Diagram *sequence* mengunduh laporan total perkara (iterasi kedua)



Gambar 5.14 Diagram *sequene* mengunduh laporan total perkara (iterasi kedua)

Diagram *sequence* mengunduh laporan total perkara pada iterasi kedua dapat dilihat pada gambar 5.14. Pada diagram *sequence* ini dijelaskan hal yang harus dilakukan aktor menekan tombol dengan ikon *print*, kemudian memilih tahun, dan menekan tombol *download*. Pada AngularJs caseController akan dipanggil *method* `downloadCaseReportClicked()`, yang selanjunya akan memanggil *method* `DownloadCaseReportFactory.get(year)` pada caseService. *Request* dari AngularJs akan diterima oleh *web service* yang kemudian akan memanggil *method* `generateCaseReport(year, request, response)` untuk membuat laporan total perkara.

5.1.2 *Entity relationship diagram* (ERD)

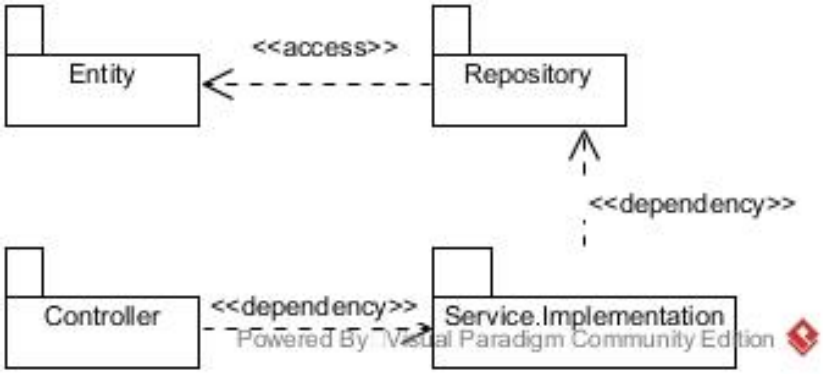


Gambar 5.15 *Entity relationship diagram*

ERD digunakan untuk memudahkan merancang basis data sebelum diimplementasikan. ERD digunakan untuk menggambarkan tabel-tabel dan kolom pada tabel yang diperlukan untuk merancang basis data. perancangan ERD berdasarkan *file excel* yang digunakan untuk mengirim data dan analisis perancangan kebutuhan dari sistem. ERD dapat dilihat pada gambar 5.15.

5.1.3 Perancangan Arsitektur

Perancangan arsitektur digunakan untuk menjelaskan arsitektur pada Spring Boot *framework*. perancangan arsitektur ini digunakan untuk menjelaskan hubungan antara arsitektur yang diimplementasikan pada penelitian dan arsitektur yang digunakan pada Spring Boot *framework*.

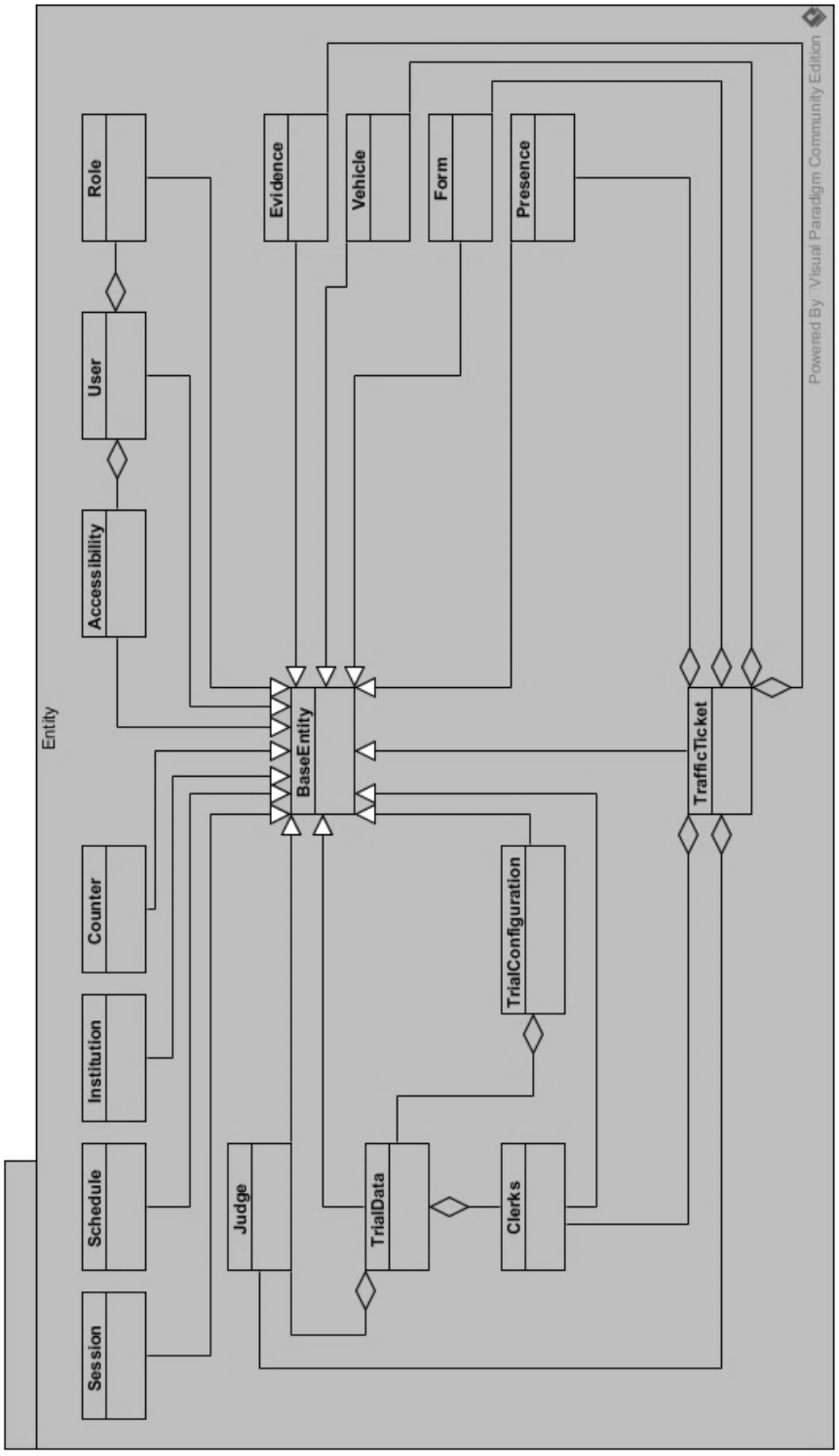


Gambar 5.16 Diagram *package* pada sistem

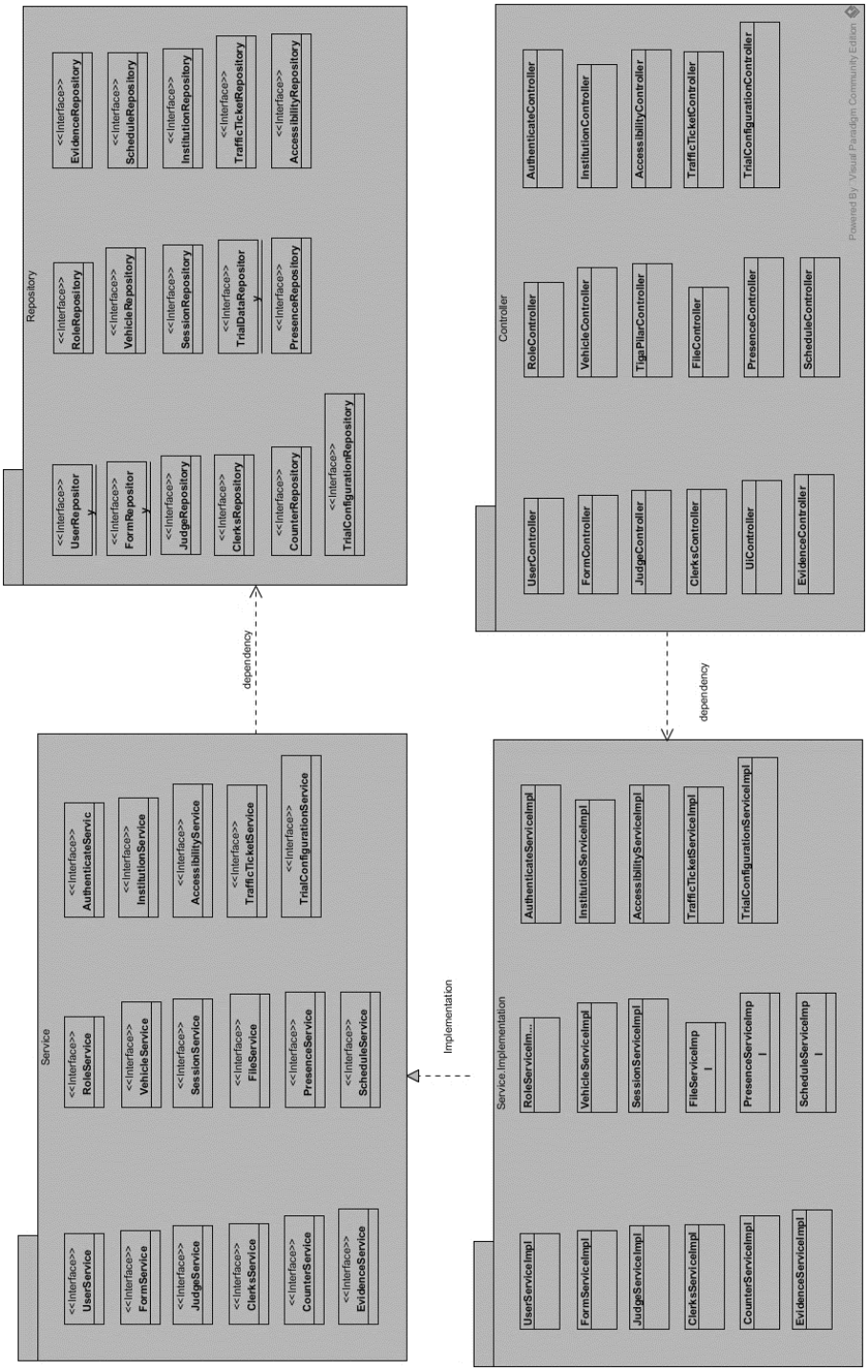
Dalam Spring Boot *framework* terdapat *web layer* yang digunakan komunikasi antara *user* dengan sistem, *service* layer yang merupakan *business logic* dari sistem, pengolahan data, dan sebagainya. *Repository layer* digunakan untuk berkomunikasi dengan basis data, dan *Domain model* merupakan objek-objek yang biasanya merepresentasikan basis data. Pada penelitian implementasi *web layer* diwakili oleh *package controller*, implementasi *service layer* diwakili oleh *package service implementation*, implementasi *repository layer* diwakili oleh *package repository,* dan implementasi *domain model* diwakili oleh *package entity*. Penjelasan detail mengenai implementasi arsitektur Spring Boot *framework* pada penelitian akan dijelaskan pada sub bab selanjutnya.

5.1.4 Diagram kelas

Diagram kelas digunakan untuk merancang kelas-kelas yang digunakan dalam implementasi sistem. Pada perancangan diagram kelas hanya berupa nama kelas yang dikelompokkan dalam beberapa *package* sesuai dengan arsitektur dari Spring Boot. Detail *attribute* serta *method* yang digunakan akan dijelaskan setelahnya dan hanya beberapa kelas dari masing-masing *package saja* yang dijelaskan yang dianggap penting dalam sistem. Perancangan diagram *class* dapat dilihat pada gambar 5.17 dan 5.18.



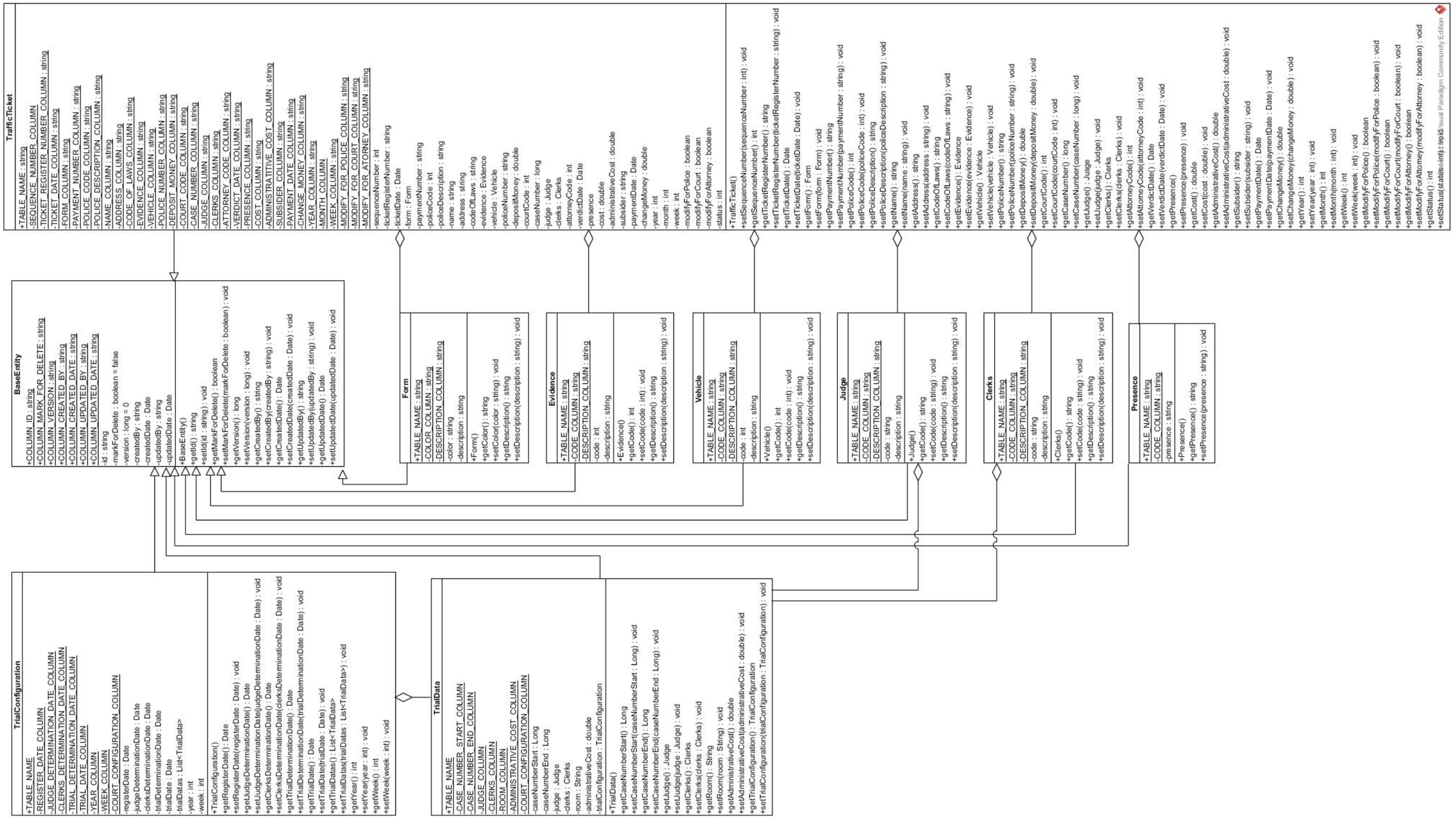
Gambar 5.17 Diagram kelas



Gambar 5.18 Diagram kelas lanjutan

Berikut merupakan penjelasan dari masing-masing *package* yang telah dibuat. Penjelasan akan dibagi menjadi lima *package* yang terdiri dari *package entity, repository, service, service implementation,* dan *controller*. Dalam penjelasan *package* akan dijelaskan fungsi utama dari kelas-kelas yang ada pada *package* tersebut.

5.1.4.1 Diagram kelas *package entity*



Gambar 5.19 Diagram Kelas *package entity*

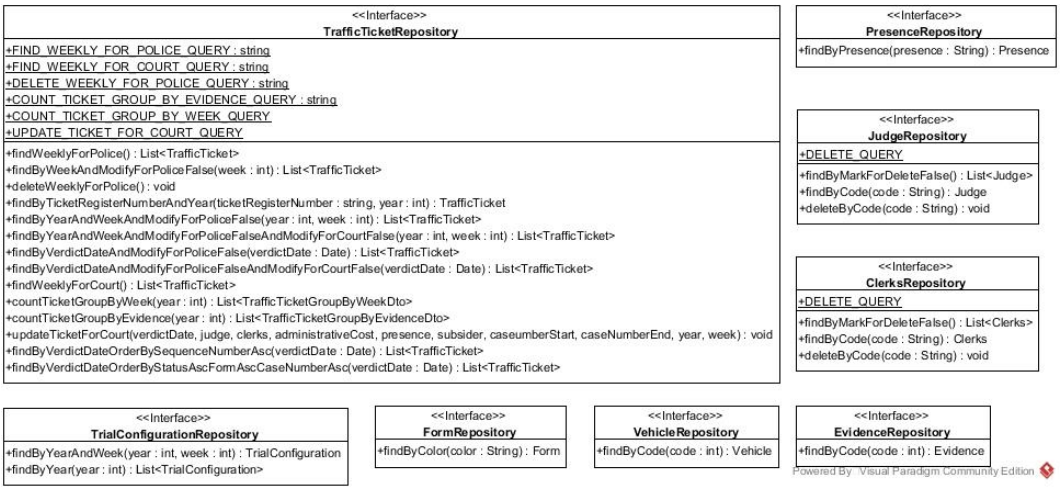
P*ackage entity* mewakili *domain model* dari Spring Boot, merupakan *package* yang akan digunakan sebagai *object relational mapping* (ORM)*,* artinya kelas-kelas pada *package entity* akan dikonversi menjadi tabel dalam basis data. Adapun kelas-kelas yang akan dijelaskan pada *package entity* ada 10 kelas, yaitu

1. kelas BaseEntity: kelas ini merupakan *parent* dari semua kelas yang ada, kelas ini digunakan untuk sebagai *auditor* dari semua data yang ada pada basis data seperti siapa yang membuat data, kapan dibuat, oleh siapa diubah, kapan diubah, dan berapa kali perubahan terjadi pada data.
2. kelas TrafficTicket: kelas ini merupakan representasi tabel data tilang pada basis data, pada kelas ini akan menyimpan terkait data tilang.
3. Kelas Form: kelas ini merupakan representasi dari tabel *form* yang ada pada basis data. Kelas ini digunakan untuk menyimpan jenis *form*.
4. Keals Evidence: kelas ini merupakan representasi dari tabel barang bukti yang ada pada basis data. Kelas ini digunakan untuk menyimpan barang bukti.
5. Kelas Vehicle: kelas ini merupakan representasi dari tabel jenis kendaraan yang ada pada basis data. kelas ini digunakan untuk menyimpan jenis kendaraan.
6. Kelas Judge: kelas ini merupakan representasi dari tabel hakim yang ada pada basis data. Kelas ini digunakan untuk menyimpan data hakim.
7. Kelas Clerks: kelas ini merupakan representasi dari tabel panitera yang ada pada basis data. Kelas ini digunakan untuk menyimpan data panitera.
8. Kelas Presence: kelas ini merupakan representasi dari tabel kehadiran terdakwa yang ada pada basis data. Kelas ini digunakan untuk menyimpan kehadiran terdakwa.
9. Kelas TrialData: kelas ini merupakan representasi dari tabel data persidangan yang ada pada basis data. Kelas ini digunakan untuk menyimpan data persidangan seperti siapa hakim dan panitera yang memutus perkara, berapa biaya perkara, dan sebagainya.
10. Kelas TrialConfiguration: kelas ini merupakan representasi dari tabel konfigurasi persidangan. Kelas ini digunakan untuk menyimpan konfigurasi persidangan seperti kapan tanggal penetapan sidang dilakukan, kapan tanggal penetapan hakim dan panitera, dan kapan tanggal sidang dilakukan.

Detail diagram kelas *package entity* dapat dilihat pada gambar 5.19.

5.1.4.2 Diagram kelas *package repository*

- **Diagram kelas *package repository* Iterasi pertama**



Gambar 5.20 Diagram kelas *package repository* iterasi pertama

Kelas-kelas pada *package repository* mewakili *repository layer* dari Spring Boot yang digunakan untuk melakukan akses ke basis data, dan akan dikonversikan kembali ke tipe data pada kelas *entity* secara *default*nya atau biasa disebut dengan *Data Access Object* (DAO). Pada Spring Boot *Framework* sendiri kelas-kelas yang digunakan untuk mengakses basis data biasanya bertipe *interface* dan meng*extends* dari *repository* yang ada pada Spring Boot *Framework* seperti JpaRepository, CrudRepository, PagingAndSortingRepository dan sebagainya. Penamaan *method-method*nya pun memiliki aturan-aturannya sendiri. pada *package repository* penulis mengambil beberapa kelas untuk dijelaskan yang dapat dilihat pada gambar 5.19.

Kelas TrafficTicketRepository digunakan untuk mengakses *entity* TrafficTicket, pada kelas ini ada *method* dengan nama `findByWeekAndModifyForPoliceFalse(week : int)` artinya *method* ini akan mencari data TrafficTicket dengan atribut `week` yang ditentukan dan atribut `modifyForPolice` dengan nilai false pada basis data. Selain itu ada *default method* seperti `save(object : T)`, `delete(id : Object)`, `count()`, dan lain sebagainya yang secara *default* tidak perlu dituliskan.

Kelas PresenceRepository digunakan untuk mengakses *entity presence*, pada kelas ini ada *method* `findByPresence(presence)` untuk mencari kehadiran berdasarkan jenis kehadirannya.

Kelas JudgeRepository digunakan untuk mengakses *entity judge*, pada kelas ini terdapat *method* `findByMarkForDeleteFalse()` untuk menampilkan data hakim dengan atribut markForDelete bernilai *false*, `findByCode(code)` untuk mencari data hakim berdasarkan kode hakim, dan `deleteByCode(code)` untuk menghapus data hakim.

Kelas ClerksRepository digunakan untuk mengakses *entity clerks*, pada kelas ini terdapat *method* `findByMarkForDeleteFalse()` untuk menampilkan data panitera dengan atribut markForDelete bernilai *false*, `findByCode(code)` untuk mencari data panitera berdasarkan kode panitera, dan `deleteByCode(code)` untuk menghapus data panitera.

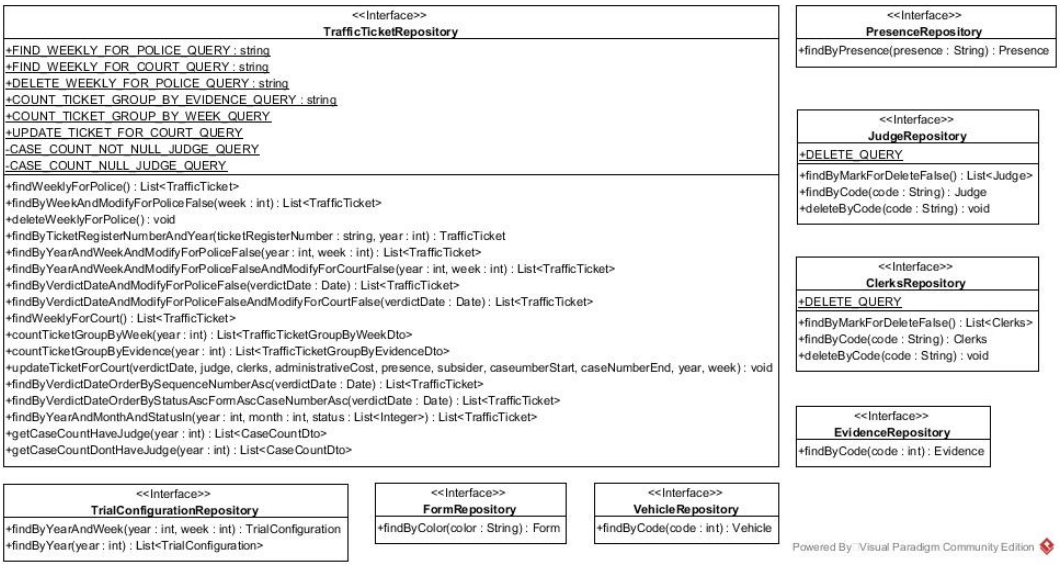
Kelas EvidenceRepository digunakan untuk mengakses *entity evidence*, pada kelas ini terdapat *method* `findByCode(code)` untuk mencari barang bukti berdasarkan kode barang bukti.

Kelas VehicleRepository digunakan untuk mengakses *entity vehicle*, pada kelas ini terdapat *method* `findByCode(code)` untuk mencari jenis kendaraan berdasarkan kode kendaraan.

Kelas FormRepository digunakan untuk mengakses *entity form*, pada kelas ini terdapat *method* `findByCode(code)` untuk mencari *form* tilang berdasarkan kode *form*.

Kelas TrialConfigurationRepository digunakan untuk mengakses *entity* trialConfiguration, pad kelas ini terdapat *method* `findByYearAndWeek(year, week)` yang digunakan untuk mencari konfigurasi persidangan pada minggu dan tahun ini. *Method* `findByYear(year)` digunakan untuk mendapatkan konfigurasi persidangan dalam satu tahun.

- **Diagram kelas *package repository* iterasi kedua**



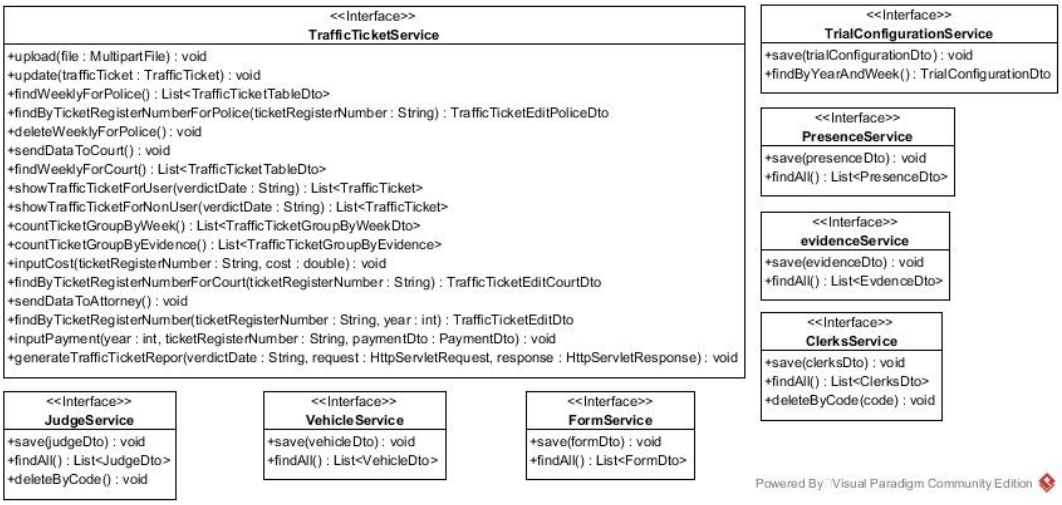
Gambar 5.21 Diagram kelas *package repository* iterasi kedua

Setelah dilakukan iterasi tahap pertama ternyata terdapat penambahan kebutuhan fungsional. Sehingga diagram kelas pada *package repository* terdapat penambahan atribut dan *method* pada kelas TrafficTicketRepository yang dapat dilihat pada gambar 5.21. Adapun penambahan atribut dan *method* tersebut yaitu:

1. CASE_COUNT_NOT_NULL_JUDGE_QUERY : *query* untuk merekap total perkara dalam satu tahun yang sudah diputus oleh hakim.
2. CASE_COUNT_NULL_JUDGE_QUERY : *query* untuk merekap total perkara dalam satu tahun yang belum diputus oleh hakim.
3. `findByYearAndMonthAndStatusIn(year, month, status)` : *method* ini digunakan untuk menampilkan data tilang dengan barang bukti yang belum diambil.
4. `getCaseCountHaveJudge(year)` : *method* ini digunakan untuk menghitung jumlah perkara tilang yang telah diputus oleh hakim.
5. `getCaseCountDontHaveJudge(year)` : *method* ini digunakan untuk menghitung jumlah perkara tilang yang belum diputus oleh hakim.

5.1.4.3 Diagram kelas *package service*

- **Diagram kelas *package service* iterasi pertama**



Gambar 5.22 Diagram kelas *package service* iterasi pertama

Package service digunakan sebagai *interface* sebelum kelas tersebut diimplementasikan. Penulis disini menggunakan *interface*, tidak langsung mengimplementasikan kelasnya karena menjaga *coupling* antar kelas, serta dengan menggunakan *interface* dapat diimplementasikan oleh kelas-kelas lain yang berbeda tanpa harus mengubah kode program. Pada *package service* penulis mengambil 8 kelas yang dianggap penting dari sistem seperti yang digambarkan pada gambar 5.22

Kelas TrafficTicketService merupakan kelas yang digunakan sebagai *interface* untuk mengurusi *business logic* dari data tilang. Pada kelas ini terdapat *method-method* utama untuk pengelolaan data tilang seperti *method* `upload(file)` yang digunakan untuk mengunggah *file excel* data tilang dan menyimpannya ke basis data, *method* `update(trafficTicket)` untuk melakukan perubahan terkait data tilang, *method* `sendDataToCourt()` untuk mengirimkan data tilang yang telah diunggah ke pengadilan, *method* `inputCost(ticketRegisterNumber, cost)` untuk memasukkan besaran denda tilang, *method* `inputPayment(year,`

`ticketRegisterNumber, paymentDto)` untuk melakukan konfirmasi pembayaran tilang, dan sebagainya.

Kelas TrialConfigurationService merupakan kelas yang digunakan sebagai *interface* untuk mengurusi *business logic* dari konfigurasi persidangan. Pada kelas ini terdapat *method* `save(trialConfigurationDto)` untuk menyimpan konfigurasi persidangan, dan *method* `findByYearAndWeek()` untuk mencari konfigurasi persidangan tahun dan minggu ini.

Kelas JudgeService merupakan kelas yang digunakan sebagai *interface* untuk mengurusi *business logic* dari data hakim. Pada kelas ini terdapat *method* `save(judgeDto)` untuk menyimpan data hakim, *method* `findAll()` untuk menampilkan data hakim, dan *method* `deleteByCode(code)` untuk menghapus data hakim.

Kelas ClerksService merupakan kelas yang digunakan sebagai *interface* untuk mengurusi *business logic* dari data panitera. Pada kelas ini terdapat *method* `save(clerksDto)` untuk menyimpan data panitera, *method* `findAll()` untuk menampilkan data panitera, dan *method* `deleteByCode(code)` untuk menghapus data panitera

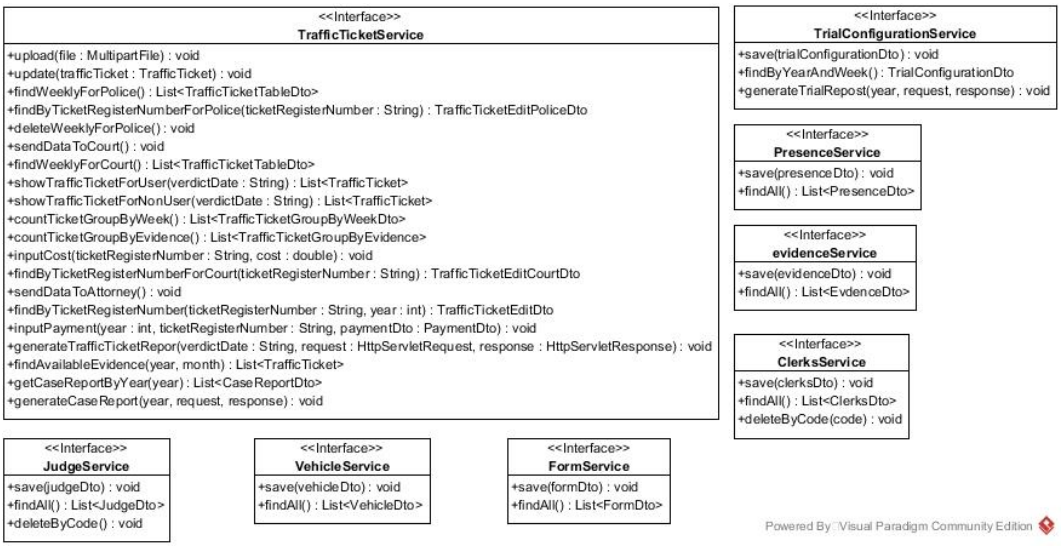
Kelas VehicleService merupakan kelas yang digunakan sebagai *interface* untuk mengurusi *business logic* dari data jenis kendaraan. Pada kelas ini terdapat *method* `save(vehicleDto)` untuk menyimpan data jenis kendaraan, *method* `findAll()` untuk menampilkan data jenis kendaraan.

Kelas EvidenceService merupakan kelas yang digunakan sebagai *interface* untuk mengurusi *business logic* dari data barang bukti. Pada kelas ini terdapat *method* `save(evidenceDto)` untuk menyimpan data barang bukti, *method* `findAll()` untuk menampilkan data barang bukti.

Kelas FormService merupakan kelas yang digunakan sebagai *interface* untuk mengurusi *business logic* dari data *form* tilang. Pada kelas ini terdapat *method* `save(formDto)` untuk menyimpan data *form* tilang, *method* `findAll()` untuk menampilkan data *form* tilang.

Kelas PresenceService merupakan kelas yang digunakan sebagai *interface* untuk mengurusi *business logic* dari data kehadiran terdakwa. Pada kelas ini terdapat *method* `save(presenceDto)` untuk menyimpan data kehadiran terdakwa, *method* `findAll()` untuk menampilkan data kehadiran terdakwa.

- **Diagram kelas *package service* iterasi kedua**



Gambar 5.23 Diagram kelas *package service* iterasi kedua

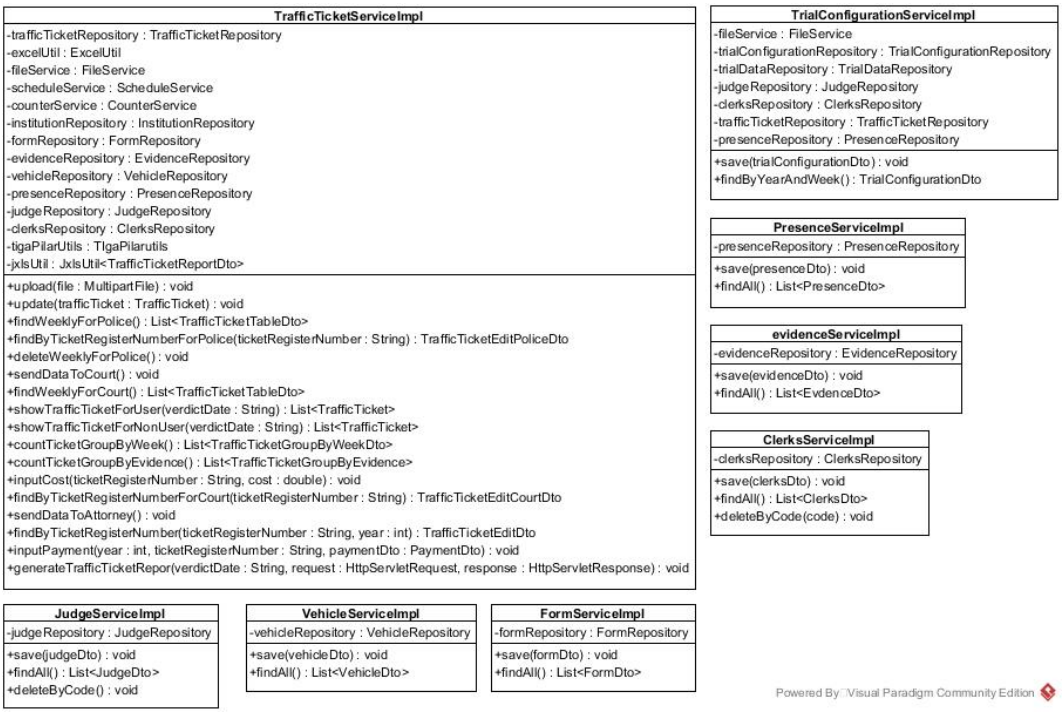
Setelah dilakukan iterasi tahap pertama, ternyata terdapat penambahan kebutuhan fungsional. Sehingga pada kelas TrafficTicketService dan kelas TrialConfigurationService terdapat penambahan beberapa *method* yang dapat dilihat pada gambar 5.23, penambahan *method* tersebut antara lain:

1. `findAvailableEvidence(year)` : *method* ini digunakan sebagai *interface* untuk menampilkan data tilang dengan barang bukti yang belum diambil.
2. `getCaseReportByYear(year)` : *method* ini digunakan sebagai *interface* untuk menampilkan rekap perkara dalam satu tahun.
3. `generateCaseReport(year, request, response)` : *method* ini digunakan sebagai *interface* untuk membuat laporan rekap perkara dalam satu tahun dalam bentuk *file excel*.
4. `generateTrialReport(year, request, response)` : *method* ini digunakan sebagai *interface* untuk membuat laporan persidangan dalam satu tahun dalam bentuk *file excel*.

5.1.4.4 Diagram kelas *package service implementation*

- **Diagram kelas *package service implementation* iterasi pertama**

Package service implementation mewakili *service layer* pada diagram arsitektur Spring Boot merupakan sebuah *package* yang didalamnya terdapat kelas-kelas mengenai *business logic* dari sistem yang diimplementasikan. *Package service implementation* merupakan implementasi dari *package service*, pada *package* ini kelas-kelas sudah diimplementasikan sesuai dengan fungsinya masing-masing. Pada *package service implementation* penulis mengambil 8 kelas yang dianggap penting. Diagram *package service implementation* dapat dilihat pada gambar 5.24.



Gambar 5.24 Diagram kelas *package service implementation* iterasi pertama

Kelas TrafficTicketServiceImpl merupakan kelas untuk mengurusi *business logic* dari data tilang. Pada kelas ini terdapat *method-method* utama untuk pengelolaan data tilang seperti *method* `upload(file)` yang digunakan untuk mengunggah *file excel* data tilang dan menyimpannya ke basis data, *method* `update(trafficTicket)` untuk melakukan perubahan terkait data tilang, *method* `sendDataToCourt()` untuk mengirimkan data tilang yang telah diunggah ke pengadilan, *method* `inputCost(ticketRegisterNumber, cost)` untuk memasukkan besaran denda tilang, *method* `inputPayment(year, ticketRegisterNumber, paymentDto)` untuk melakukan konfirmasi pembayaran tilang, dan sebagainya.

Kelas TrialConfigurationServiceImpl merupakan kelas untuk mengurusi *business logic* dari konfigurasi persidangan. Pada kelas ini terdapat *method* `save(trialConfigurationDto)` untuk menyimpan konfigurasi persidangan, dan *method* `findByYearAndWeek()` untuk mencari konfigurasi persidangan tahun dan minggu ini.

Kelas JudgeServiceImpl merupakan kelas untuk mengurusi *business logic* dari data hakim. Pada kelas ini terdapat *method* `save(judgeDto)` untuk menyimpan data hakim, *method* `findAll()` untuk menampilkan data hakim, dan *method* `deleteByCode(code)` untuk menghapus data hakim.

Kelas ClerksServiceImpl merupakan kelas untuk mengurusi *business logic* dari data panitera. Pada kelas ini terdapat *method* `save(clerksDto)` untuk menyimpan data panitera, *method* `findAll()` untuk menampilkan data panitera, dan *method* `deleteByCode(code)` untuk menghapus data panitera

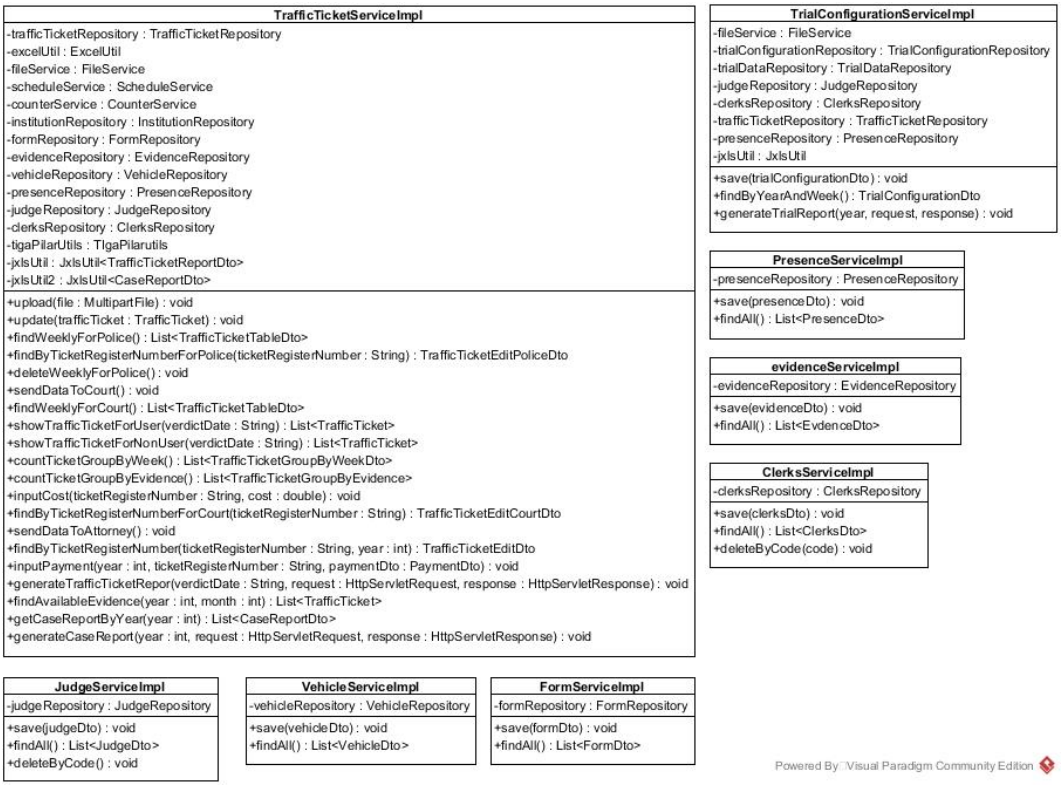
Kelas VehicleServiceImpl merupakan kelas untuk mengurusi *business logic* dari data jenis kendaraan. Pada kelas ini terdapat *method* `save(vehicleDto)` untuk menyimpan data jenis kendaraan, *method* `findAll()` untuk menampilkan data jenis kendaraan.

Kelas EvidenceServiceImpl merupakan kelas untuk mengurusi *business logic* dari data barang bukti. Pada kelas ini terdapat *method* `save(evidenceDto)` untuk menyimpan data barang bukti, *method* `findAll()` untuk menampilkan data barang bukti.

Kelas FormServiceImpl merupakan kelas untuk mengurusi *business logic* dari data *form* tilang. Pada kelas ini terdapat *method* `save(formDto)` untuk menyimpan data *form* tilang, *method* `findAll()` untuk menampilkan data *form* tilang.

Kelas PresenceServiceImpl merupakan kelas untuk mengurusi *business logic* dari data kehadiran terdakwa. Pada kelas ini terdapat *method* `save(presenceDto)` untuk menyimpan data kehadiran terdakwa, *method* `findAll()` untuk menampilkan data kehadiran terdakwa.

- **Diagram kelas pada *package service implementation* iterasi kedua**



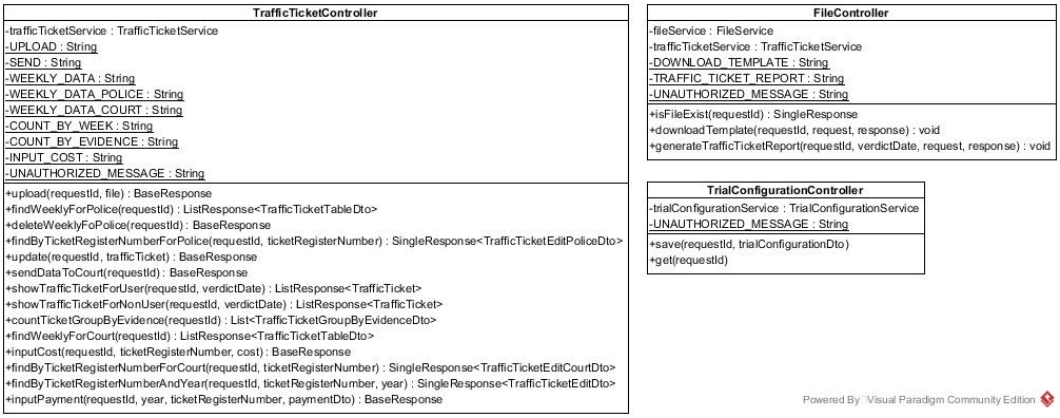
Gambar 5.25 Diagram kelas *package service implementation* iterasi kedua

Setelah dilakukan iterasi tahap pertama, ternyata terdapat penambahan kebutuhan fungsional. Sehingga pada kelas TrafficTicketServiceImpl dan TrialConfigurationServiceImpl terdapat penambahan beberapa *method* yang dapat dilihat pada gambar 5.25, penambahan *method* tersebut antara lain:

1. `findAvailableEvidence(year)` : *method* ini digunakan untuk menampilkan data tilang dengan barang bukti yang belum diambil.
2. `getCaseReportByYear(year)` : *method* ini digunakan untuk menampilkan rekap perkara dalam satu tahun.
3. `generateCaseReport(year, request, response)` : *method* ini digunakan untuk membuat laporan rekap perkara dalam satu tahun dalam bentuk *file excel*.
4. `generateTrialReport(year, request, response)` : *method* ini digunakan untuk membuat laporan persidangan dalam satu tahun dalam bentuk *file excel*.

5.1.4.5 Diagram kelas *package controller*

- **Diagram kelas *package controller* iterasi pertama**



Gambar 5.26 Diagram kelas *package controller* iterasi pertama

Package controller mewakili *web layer* Spring Boot merupakan *package* yang didalamnya terdapat kelas-kelas yang dibuat sebagai *web service* menggunakan *Uniform Resource Identifier* (URI). Pada *package controller* penulis mengambil 3 kelas utama dari sistem. Diagram kelas *package controller* iterasi pertama dapat dilihat pada gambar 5.26.

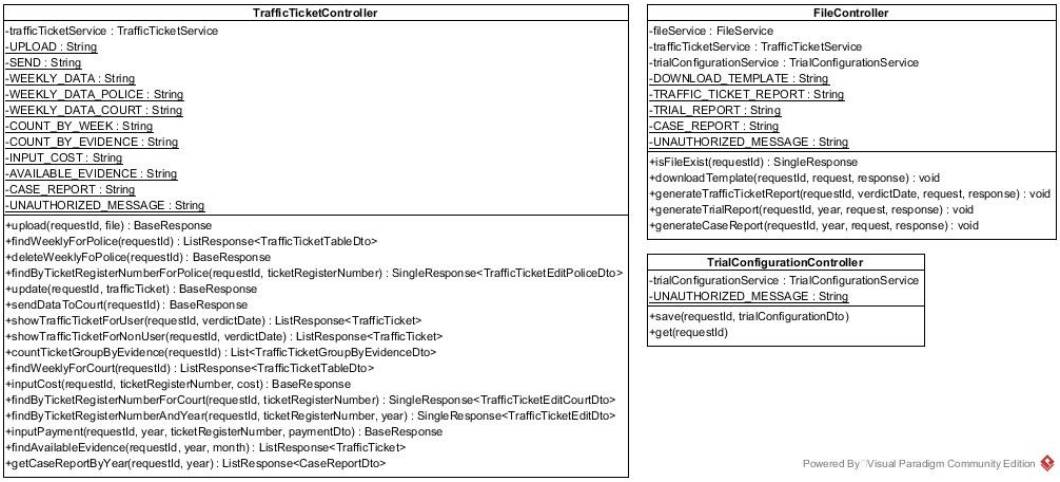
Kelas TrafficTicketController digunakan sebagai API *path* dari *web service* untuk mengakses data tilang. Pada kelas ini terdapat *method-method* utama seperti *method* `upload(requestId, file)` yang sebagai API *path web service* untuk mengunggah *file excel* data tilang, *method* `sendDataToCourt(requestId)` yang digunakan sebagai API *path web service* untuk mengirimkan data tilang ke pengadilan, *method* `inputCost(requestId, ticketRegisterNumber, cost)` digunakan sebagai API *path web service* untuk memasukkan denda tilang, *method* `inputPayment(requestId, ticketRegisterNumber, paymentDto)` digunakan sebagai API *path web service* untuk konfirmasi pembayaran tilang.

Kelas TrialConfigurationController digunakan sebagai API *path* dari *web service* untuk mengakses konfigurasi persidangan. Pada kelas ini terdapat *method* `save(requestId,trialConfigurationDto)` digunakan sebagai API *path web*

service untuk menyimpan konfigurasi persidangan, dan *method* `get(requestId)` digunakan sebagai API *path web service* untuk menampilkan konfigurasi persidangan minggu ini.

Kelas FileController digunakan sebagai API *path* dari *web service* untuk mengakses *file* atau membuat *file* laporan. Pada kelas ini terdapat *method* `generateTrafficReport(requestId, verdictDate, request, response)` digunakan sebagai API *path web service* untuk membuat laporan data tilang.

- **Diagram kelas *package controller* iterasi kedua**



Gambar 5.27 Diagram kelas *package controller* iterasi kedua

Pada *package* controller terdapat perubahan pada iterasi kedua, hal ini dikarenakan terdapat penambahan kebutuhan fungsional dari *user*. Kelas yang berubah pada *package controller* adalah kelas TrafficTicketController dan FileController yang dapat dilihat pada gambar 5.27. Beberapa penambahan *method-method* pada kelas tersebut antara lain:

1. `findAvailableEvidence(requestId, year, month)` : *method* ini digunakan sebagai API *path web service* untuk menampilkan data tilang yang belum diambil barang buktinya.
2. `getCaseReportByYear(requestId, year)` : *method* ini digunakan sebagai API *path web service* untuk menampilkan rekap total perkara tilang dalam satu tahun.
3. `generateTrialReport(requestId, year, request, response)` : *method* ini digunakan sebagai API *path web service* untuk mengunduh laporan persidangan dalam satu tahun.
4. `generateCaseReport(requestId, year, request, response)` : *method* ini digunakan sebagai API *path web service* untuk mengunduh laporan rekap perkara tilang dalam satu tahun.

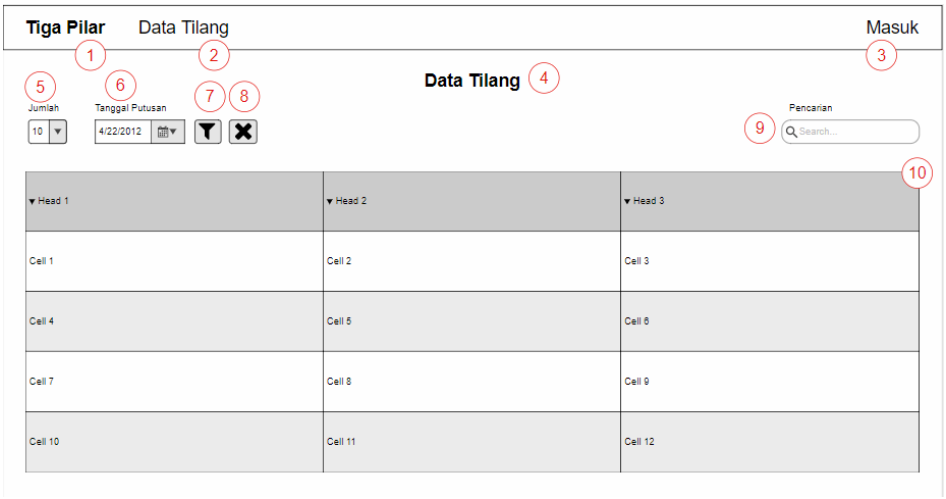
5.1.5 Perancangan antarmuka

Pada perancangan antarmuka akan ditunjukkan bagaimana rancangan antarmuka yang ditampilkan untuk mempermudah interaksi antara pengguna dan sistem. Pada tahap perancangan antarmuka alat bantu yang digunakan untuk membuat adalah *Moqups* yang diakses secara daring.

- **Perancangan antarmuka iterasi pertama**

5.1.5.1 Perancangan antarmuka halaman *user* - data tilang

Antarmuka halaman *user* – data tilang digunakan untuk menampilkan data tilang setiap minggunya kepada masyarakat serta sebagai halaman *default* ketika mengakses sistem informasi tiga pilar. Pada halaman ini terdapat tabel untuk menampilkan data tilang, serta adanya pencarian dan *filtering* yang memudahkan masyarakat untuk mencari datanya. Gambar perancangan antarmuka ditunjukkan pada gambar 5.28 dan penjelasan dari perancangan antarmuka dijelaskan pada tabel 5.1.



Gambar 5.28 Gambar perancangan antramuka halaman *user* – data tilang

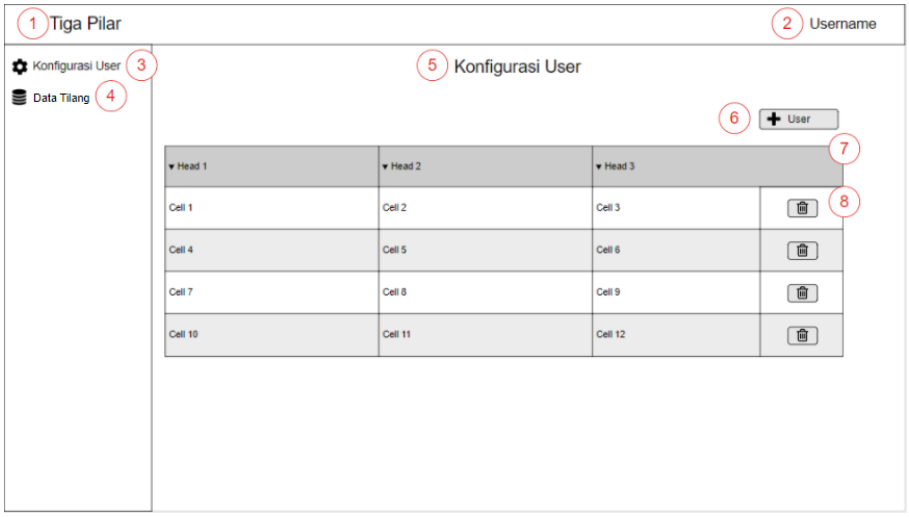
Tabel 5.1 Penjelasan perancangan antarmuka halaman *user* – data tilang

No	Nama	Tipe	Deskripsi
1	Tiga Pilar	Teks	Nama dari sistem.
2	Data Tilang	Link	*Link* untuk mengakses halama data tilang.
3	Masuk	Teks	Teks untuk membuka *popup* autentikasi untuk masuk kedalam sistem.
4	Judul	Teks	Judul dari halaman saat ini.
5	Jumlah	*Dropdown*	*Dropdown* untuk menampilkan berapa jumlah data yang ditampilkan pada tabel.
6	Tanggal Putusan	*Input*	*Input* untuk memilih tanggal putusan untuk melakukan *filtering* data.
7	*Filter*	Tombol	Tombol untuk melakukan *filtering* data.

8	*Clear*	Tombol	Tombol untuk menghapus *filtering* data dan menampilkan data pada minggu ini.
9	Pencarian	*Input*	*Input* teks untuk melakukan pencarian data
11	Tabel data tilang	Tabel	Tabel untuk menampilkan data tilang.

5.1.5.2 Perancangan antarmuka halaman *administrator* – konfigurasi

Pada halaman *administrator* – konfigurasi merupakan halaman yang dibuka ketika aktor telah melakukan autentikasi dan memiliki *role administrator*. Halaman ini digunakan oleh aktor untuk melihat siapa saja yang memiliki akses kedalam sistem, termasuk penambahan *user*, ataupun penghapusan *user* yang telah ada. Gambar perancangan antarmuka ditunjukkan pada gambar 5.29 dan penjelasan dari perancangan antarmuka dijelaskan pada tabel 5.2.



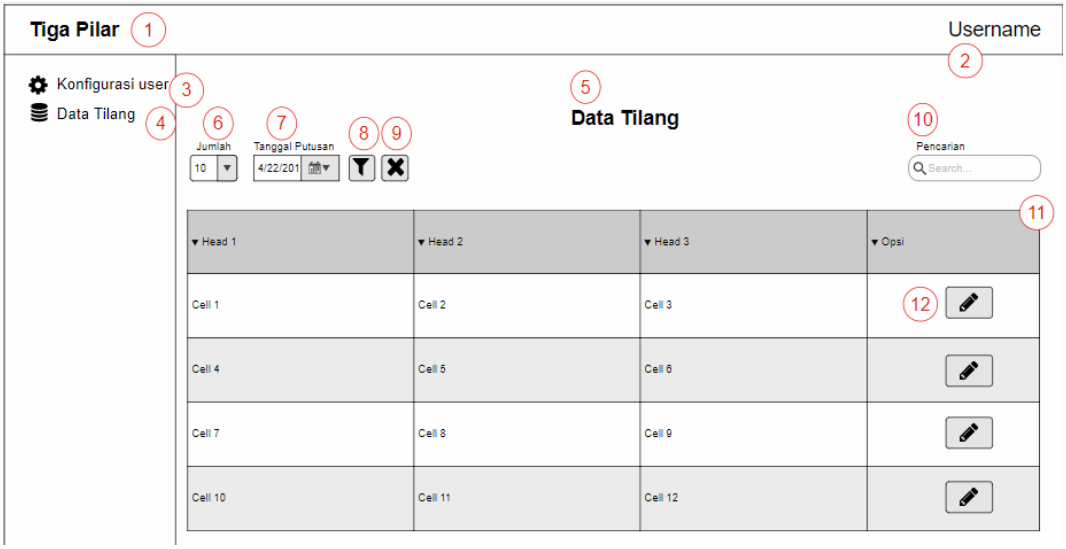
Gambar 5.29 Gambar perancangan antarmuka halaman *administrator* – konfigurasi

Tabel 5.2 Penjelasan perancangan antarmuka halaman *administrator* – konfigurasi

No	Nama	Tipe	Deskripsi
1	Tiga Pilar	Teks	Nama dari sistem.
2	Username	*Dropdown*	Nama dari *user* yang telah masuk.
3	Menu konfigurasi	*Link*	Menu untuk masuk ke halaman konfigurasi.
4	Menu Data Tilang	*Link*	Menu untuk masuk ke halaman data tilang.
5	*Header*	Teks	Judul dari halaman.
6	Tambah user	Tombol	Tombol untuk menambahkan *user* baru.
7	Tabel user	Tabel	Tabel untuk menampilkan *user*.
8	hapus	Tombol	Tombol untuk menghapus *user*.

5.1.5.3 Perancangan antarmuka halaman *administrator* – data tilang

Pada halaman *administrator* – data tilang merupakan halaman yang digunakan oleh aktor untuk melihat data tilang per minggunya, selain itu aktor juga dapat melakukan *filtering* data yang ingin ditampilkan termasuk melakukan pengubahan data jika diperlukan. Gambar perancangan antarmuka ditunjukkan pada gambar 5.30 dan penjelasan dari perancangan antarmuka dijelaskan pada tabel 5.3.



Gambar 5.30 Gambar perancangan antarmuka halaman *administrator* – data tilang

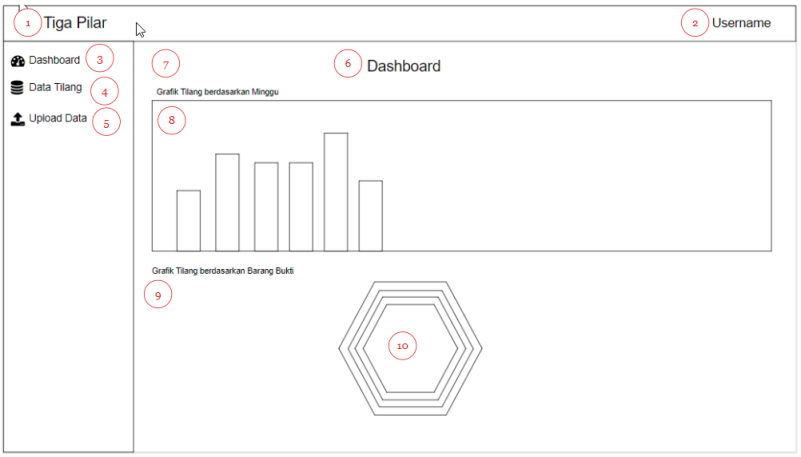
Tabel 5.3 Penjelasan perancangan antarmuka halaman *administrator* – data tilang

No	Nama	Tipe	Deskripsi
1	Tiga Pilar	Teks	Nama dari sistem.
2	Username	*Dropdown*	Nama dari *user* yang telah masuk.
3	Menu konfigurasi	*Link*	Menu untuk masuk ke halaman konfigurasi.
4	Menu data tilang	*Link*	Menu untuk masuk ke halaman data tilang.
5	*Header*	Teks	Judul dari halaman.
6	Jumlah	*Dropdown*	*Dropdown* untuk menampilkan berapa jumlah data yang ditampilkan pada tabel.
7	Tanggal Putusan	*Input*	*Input* untuk memilih tanggal putusan untuk melakukan *filtering* data.
8	*Filter*	Tombol	Tombol untuk melakukan *filtering* data.
9	*Clear*	Tombol	Tombol untuk menghapus *filtering* data dan menampilkan data pada minggu ini.
10	Pencarian	*Input*	*Input* teks untuk melakukan pencarian data
11	Tabel data tilang	Tabel	Tabel untuk menampilkan data tilang.

12	Edit	Tombol	Tombol untuk mengedit data tilang.

5.1.5.4 Perancangan antarmuka halaman kepolisian – *dashboard*

Halaman kepolisian – *dashboard* ditampilkan awal ketika aktor telah melakukan autentikasi pada sistem dan memiliki *role* kepolisian. Pada halaman ini ditampilkan statistik data tilang berdasarkan minggu dan barang bukti pelanggaran pada bentuk grafik untuk memudahkan pembacaan data. Gambar perancangan antarmuka ditunjukkan pada gambar 5.31 dan penjelasan dari perancangan antarmuka dijelaskan pada tabel 5.4.



Gambar 5.31 Gambar perancangan antarmuka halaman kepolisian - *dashboard*

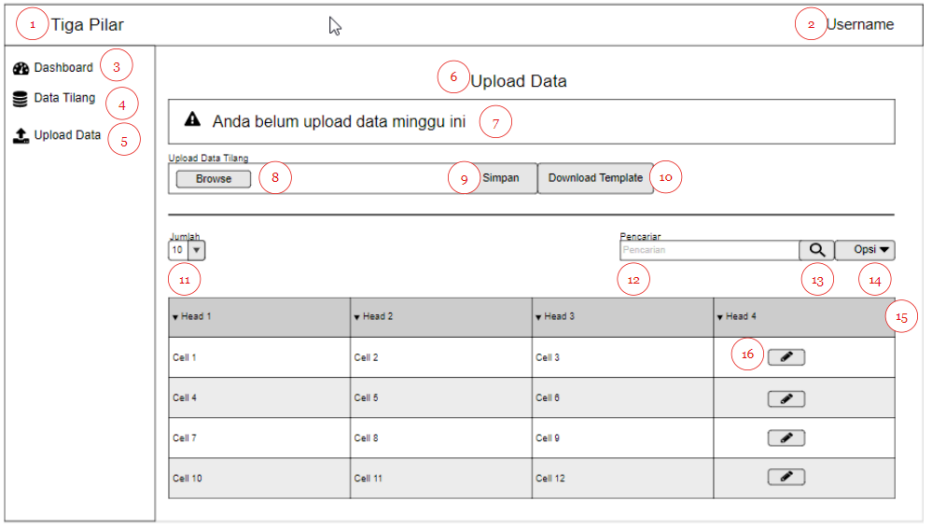
Tabel 5.4 Penjelasan perancangan antarmuka halaman kepolisian - *dashboard*

No	Nama	Tipe	Deskripsi
1	Tiga Pilar	Teks	Nama dari sistem.
2	Username	*Dropdown*	Nama dari *user* yang telah masuk.
3	Menu dashboard	*Link*	Menu untuk masuk ke halaman *dashboard*.
4	Menu data tilang	*Link*	Menu untuk masuk ke halaman data tilang.
5	Menu *Upload* data	*Link*	Menu untuk masuk ke halaman *upload* data.
6	*Header*	Teks	Judul dari halaman.
7	Judul chart	Teks	Nama dari *chart*
8	Data Tilang per Minggu	*Bar Chart*	*Chart* yang menampilkan jumlah data tilang per minggu berdasarkan tahun.
9	Judul Chart	Teks	Nama dari *chart.*
10	Data Tilang berdasarkan Barang Bukti	*Radar Chart*	*Chart* yang menampilkan jumlah data tilang berdasarkan barang bukti.

5.1.5.5 Perancangan antarmuka halaman kepolisian – *upload data*

Halaman kepolisian – *upload* data digunakan oleh aktor kepolisian untuk menggunggah data tilang setiap minggunya pada *file excel*. Di halaman ini terdapat

fitur untuk mengunduh *template* laporan dan menggunggahnya, ketika terjadi kesalahan data tilang, aktor dapat membenarkannya dengan fitur *edit* data tilang, setelah data yang diunggah telah benar maka aktor dapat mengirimkannya untuk diproses oleh pengadilan. Gambar perancangan antarmuka ditunjukkan pada gambar 5.32 dan penjelasan dari perancangan antarmuka dijelaskan pada tabel 5.5.



Gambar 5.32 Gambar perancangan antarmuka halaman kepolisian – *upload* data

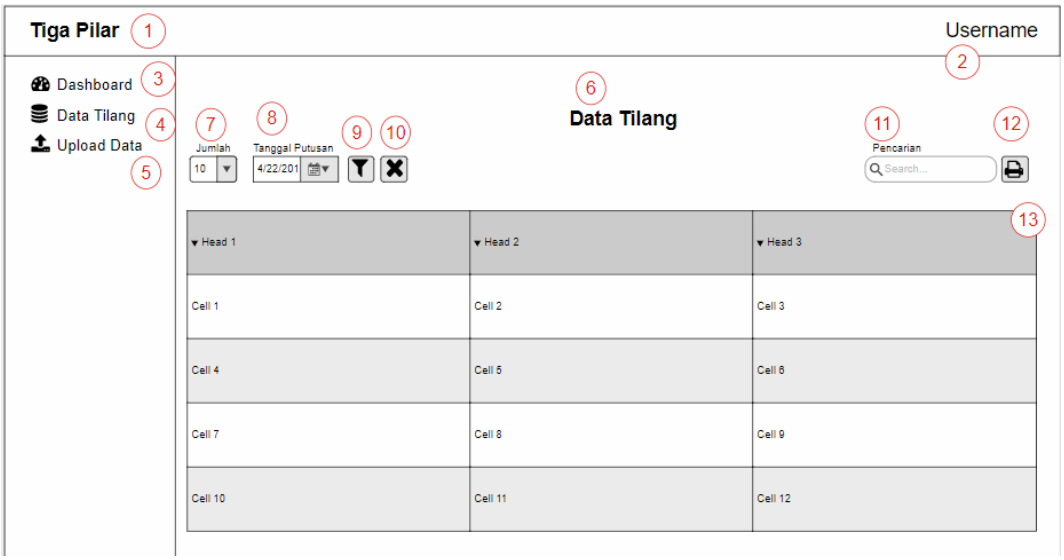
Tabel 5.5 Penjelasan perancangan antarmuka halaman kepolisian – *upload* data

No	Nama	Tipe	Deskripsi
1	Tiga Pilar	Teks	Nama dari sistem.
2	Username	*Dropdown*	Nama dari *user* yang telah masuk.
3	Menu dashboard	*Link*	Menu untuk masuk ke halaman *dashboard*.
4	Menu data tilang	*Link*	Menu untuk masuk ke halaman data tilang.
5	Menu *Upload* data	*Link*	Menu untuk masuk ke halaman *upload* data.
6	*Header*	Teks	Judul dari halaman.
7	Notifikasi	Teks	Notifikasi jika pengguna belum atau sudah menggunggah *file*.
8	File Tilang	*Input File*	*Input* untuk memasukkan file data tilang
9	Simpan	Tombol	Tombol untuk menggunggah data.
10	Unduh *Template*	Tombol	Tombol untuk mengunduh *template file* data tilang.
11	Jumlah	*Dropdown*	Untuk menampilkan berapa banyak data tilang yang ditampilkan.
12	Pencarian	*Input*	Untuk melakukan pencarian data tilang berdasarkan kata kunci yang dimasukkan.
13	*Search*	Tombol	Untuk melakukan pencarian

14	Opsi	Tombol *Dropdown*	Tombol pilihan untuk data tilang.
15	Tabel data tilang	Tabel	Tabel untuk menampilkan data tilang yang telah diunggah.
16	*Edit*	Tombol	Tombol untuk mengubah data tilang.

5.1.5.6 Perancangan antarmuka halaman kepolisian – data tilang

Pada halaman kepolisian – data tilang digunakan untuk menampilkan data tilang yang berdasarkan tahun dan minggu. Disini terdapat fitur *filtering* data dan pencarian untuk memudahkan aktor, serta jika aktor ingin mencetak laporan, dapat mengunduh *file excel* baru kemudian dicetak. Gambar perancangan antarmuka ditunjukkan pada gambar 5.33 dan penjelasan dari perancangan antarmuka dijelaskan pada tabel 5.6.



Gambar 5.33 Gambar perancangan antarmuka halaman kepolisian – data tilang

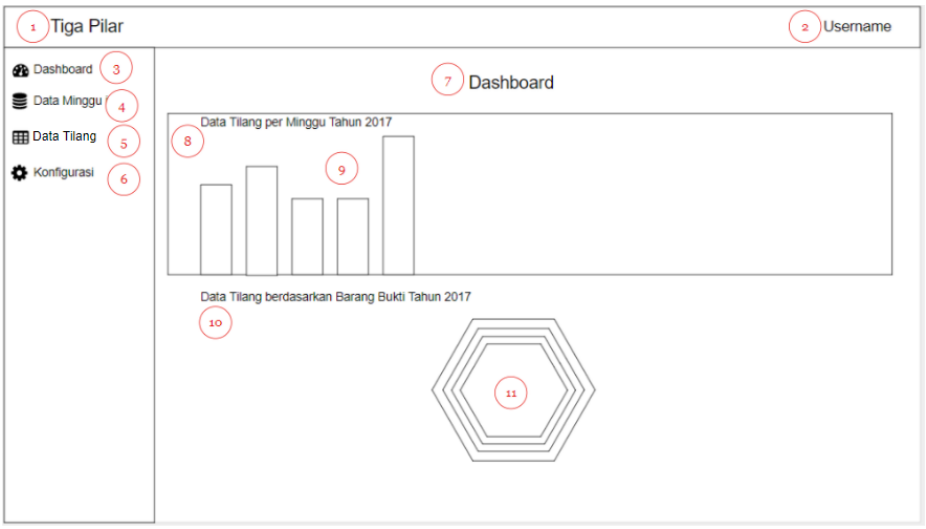
Tabel 5.6 Penjelasan perancangan antarmuka halaman kepolisian – data tilang

No	Nama	Tipe	Deskripsi
1	Tiga Pilar	Teks	Nama dari sistem.
2	Username	*Dropdown*	Nama dari *user* yang telah masuk.
3	Menu dashboard	*Link*	Menu untuk masuk ke halaman *dashboard*.
4	Menu data tilang	*Link*	Menu untuk masuk ke halaman data tilang.
5	Menu *Upload* data	*Link*	Menu untuk masuk ke halaman *upload* data.
6	*Header*	Teks	Judul dari halaman.
7	Jumlah	*Dropdown*	*Dropdown* untuk menampilkan berapa jumlah data yang ditampilkan pada tabel.
8	Tanggal Putusan	*Input*	*Input* untuk memilih tanggal putusan untuk melakukan *filtering* data.
9	*Filter*	Tombol	Tombol untuk melakukan *filtering* data.

10	*Clear*	Tombol	Tombol untuk menghapus *filtering* data dan menampilkan data pada minggu ini.
11	Pencarian	*Input*	*Input* teks untuk melakukan pencarian data
12	Unduh	Tombol	Tombol yang digunakan untuk mengunduh laporan tilang dalam bentuk *excel.*
13	Tabel data tilang	Tabel	Tabel untuk menampilkan data tilang.

5.1.5.7 Perancangan antarmuka halaman pengadilan – *dashboard*

Halaman pengadilan – dashboard ditampilkan awal ketika aktor telah melakukan autentikasi pada sistem dan memiliki *role* pengadilan. Pada halaman ini ditampilkan statistik data tilang berdasarkan minggu dan barang bukti pelanggaran pada bentuk grafik untuk memudahkan pembacaan data. Gambar perancangan antarmuka ditunjukkan pada gambar 5.34 dan penjelasan dari perancangan antarmuka dijelaskan pada tabel 5.7.



Gambar 5.34 Gambar perancangan antarmuka halaman pengadilan – *dashboard*

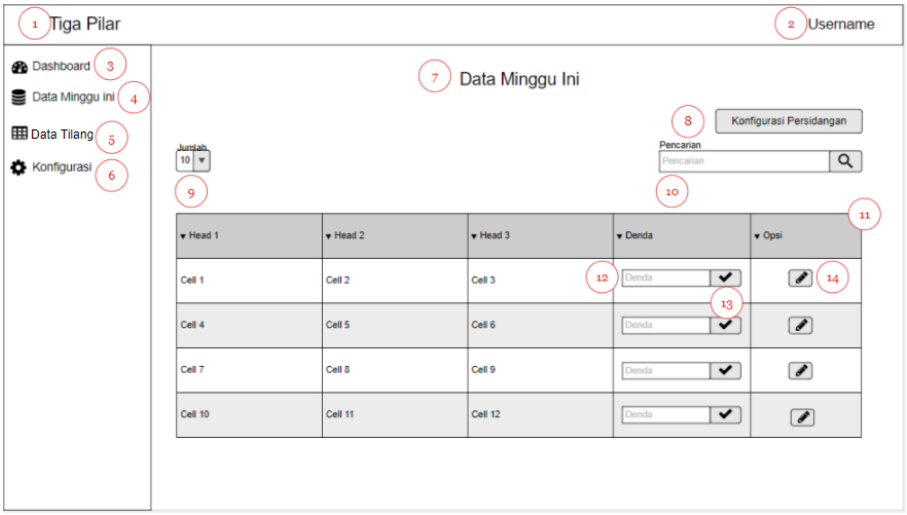
Tabel 5.7 Penjelasan perancangan antarmuka halaman pengadilan - *dashboard*

No	Nama	Tipe	Deskripsi
1	Tiga Pilar	Teks	Nama dari sistem.
2	Username	*Dropdown*	Nama dari *user* yang telah masuk.
3	Menu *dashboard*	*Link*	Menu untuk masuk ke halaman *dashboard*.
4	Menu data minggu ini	*Link*	Menu untuk masuk ke halaman data minggu ini.
5	Menu data tilng	*Link*	Menu untuk masuk ke halaman data tilang.
6	Menu konfigurasi	*Link*	Menu untuk masuk ke halaman konfigurasi.
7	*Header*	Teks	Judul dari halaman.
8	Judul chart	Teks	Nama dari *chart*

9	Data Tilang per Minggu	*Bar Chart*	*Chart* yang menampilkan jumlah data tilang per minggu berdasarkan tahun.
10	Judul Chart	Teks	Nama dari *chart.*
11	Data Tilang berdasarkan Barang Bukti	*Radar Chart*	*Chart* yang menampilkan jumlah data tilang berdasarkan barang bukti.

5.1.5.8 Perancangan antarmuka halaman pengadilan – data minggu ini

Pada halaman pengadilan – data minggu ini merupakan halaman untuk menampilkan data tilang minggu ini sebelum disidangkan, di halaman ini terdapat fitur untuk melakukan konfigurasi persidangan, memasukkan denda yang telah diputus oleh hakim dan, mengedit data tilang jika terjadi kesalahan. Gambar perancangan antarmuka ditunjukkan pada gambar 5.35 dan penjelasan dari perancangan antarmuka dijelaskan pada tabel 5.8.



Gambar 5.35 Gambar perancangan antarmuka halaman pengadilan – data minggu ini

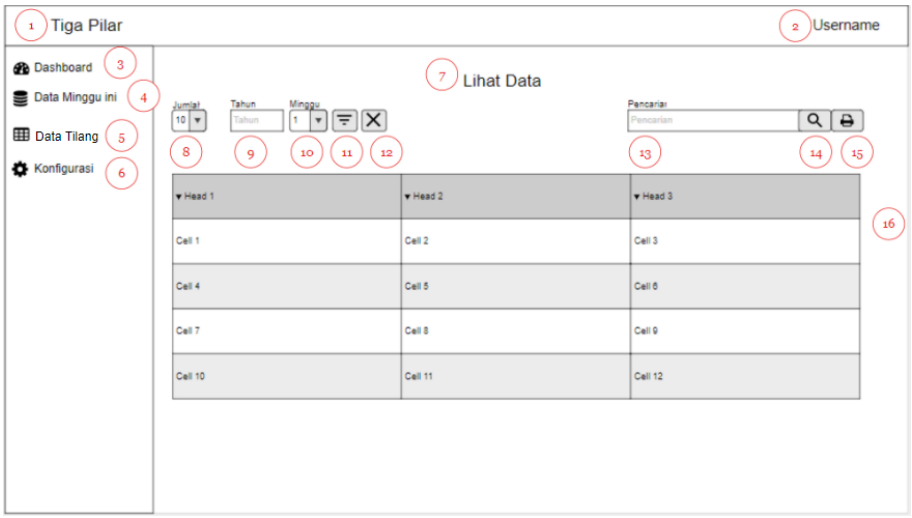
Tabel 5.8 Penjelasan perancangan antarmuka halaman pengadilan – data minggu ini

No	Nama	Tipe	Deskripsi
1	Tiga Pilar	Teks	Nama dari sistem.
2	Username	*Dropdown*	Nama dari *user* yang telah masuk.
3	Menu *dashboard*	*Link*	Menu untuk masuk ke halaman *dashboard*.
4	Menu data minggu ini	*Link*	Menu untuk masuk ke halaman data minggu ini.
5	Menu data tilang	*Link*	Menu untuk masuk ke halaman data tilang.
6	Menu konfigurasi	*Link*	Menu untuk masuk ke halaman konfigurasi.
7	*Header*	Teks	Judul dari halaman.

8	Konfigurasi persidangan	Tombol	Tombol untuk memasukkan konfigurasi persidangan.
9	Jumlah	*Dropdown*	Untuk memilih berapa banyak data tilang yang ditampilkan pada satu halaman.
10	Pencarian	*Input*	Untuk melakukan pencarian data tilang.
11	Tabel data tilang	Tabel	Tabel untuk menampilkan data tilang.
12	*Input* denda	*Input*	*Input* untuk memasukkan denda tilang yang telah diputus.
13	*Submit* denda	Tombol	Tombol untuk mengirimkan dendan yang telah dimasukkan.
14	*edit*	Tombol	Tombol untuk melakukan pengubahan data tilang.

5.1.5.9 Perancangan antarmuka halaman pengadilan – data tilang

Pada halaman pengadilan – data tilang digunakan untuk menampilkan data tilang berdasarkan tahun dan minggu. Gambar perancangan antarmuka ditunjukkan pada gambar 5.36 dan penjelasan dari perancangan antarmuka dijelaskan pada tabel 5.9.



Gambar 5.36 Gambar perancangan antarmuka halaman pengadilan – data tilang

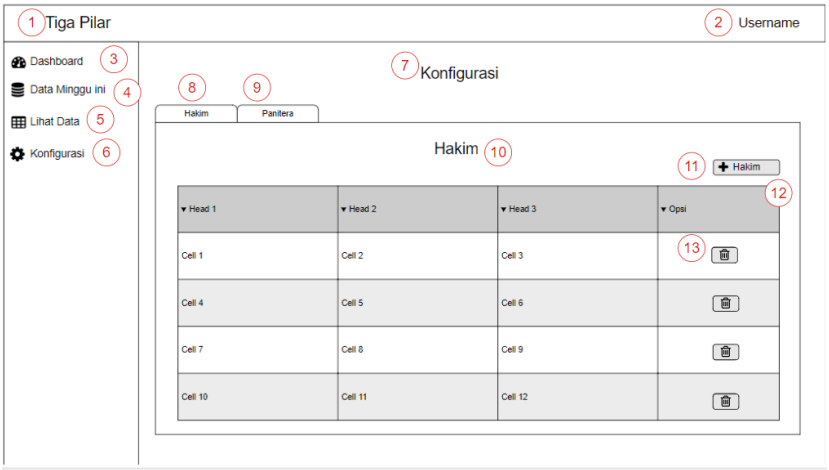
Tabel 5.9 Penjelasan perancangan antarmuka halaman pengadilan – data tilang

No	Nama	Tipe	Deskripsi
1	Tiga Pilar	Teks	Nama dari sistem.
2	Username	*Dropdown*	Nama dari *user* yang telah masuk.
3	Menu *dashboard*	*Link*	Menu untuk masuk ke halaman *dashboard*.
4	Menu data minggu ini	*Link*	Menu untuk masuk ke halaman data minggu ini.
5	Menu data tilang	*Link*	Menu untuk masuk ke halaman data tilang.

6	Menu konfigurasi	*Link*	Menu untuk masuk ke halaman konfigurasi.
7	*Header*	Teks	Judul dari halaman.
8	Jumlah	*Dropdown*	*Dropdown* untuk menampilkan berapa jumlah data yang ditampilkan pada tabel.
9	Tahun	*Input*	*Input* untuk memilih tahun untuk melakukan *filtering* data.
10	Minggu	*Dropdown*	*Dropdown* untuk memilih minggu untuk melakukan *filtering* data.
11	*Filter*	Tombol	Tombol untuk melakukan *filtering* data.
12	*Clear*	Tombol	Tombol untuk menghapus *filtering* data dan menampilkan data pada minggu ini.
13	Pencarian	*Input*	*Input* teks untuk melakukan pencarian data
14	*Search*	Tombol	Tombol untuk melakukan pencarian data
15	Unduh	Tombol	Tombol yang digunakan untuk mengunduh laporan tilang dalam bentuk *excel.*
16	Tabel data tilang	Tabel	Tabel untuk menampilkan data tilang.

5.1.5.10 *Perancangan antarmuka halaman pengadilan – konfigurasi*

Pada halaman pengadilan – konfigurasi digunakan untuk melakukan konfigurasi hakim dan panitera yang ada pada pengadilan. Gambar perancangan antarmuka ditunjukkan pada gambar 5.37 dan penjelasan dari perancangan antarmuka dijelaskan pada tabel 5.10.



Gambar 5.37 Gambar perancangan antarmuka halaman pengadilan – konfigurasi

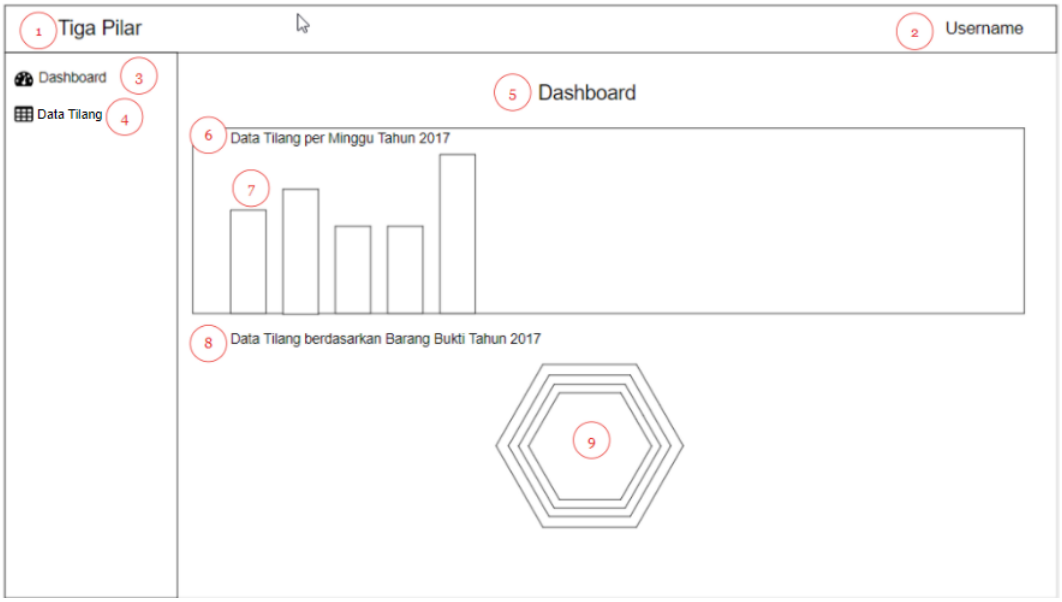
Tabel 5.10 Penjelasan perancangan antarmuka halaman pengadilan – konfigurasi

No	Nama	Tipe	Deskripsi
1	Tiga Pilar	Teks	Nama dari sistem.

2	Username	*Dropdown*	Nama dari *user* yang telah masuk.
3	Menu *dashboard*	*Link*	Menu untuk masuk ke halaman *dashboard*.
4	Menu data minggu ini	*Link*	Menu untuk masuk ke halaman data minggu ini.
5	Menu data tilang	*Link*	Menu untuk masuk ke halaman data tilang.
6	Menu konfigurasi	*Link*	Menu untuk masuk ke halaman konfigurasi.
7	*Header*	Teks	Judul dari halaman.
8	*Tab* Hakim	*Dropdown*	*Tab* untuk menampilkan *tab* hakim.
9	*Tab* Panitera	*Input*	*Tab* untuk menampilkan *tab* panitera.
10	Judul *tab*	*Dropdown*	Judul dari *tab*.
11	Tambah hakim	Tombol	Tombol untuk menambah hakim.
12	Tabel Hakim	Tombol	Tabel untuk menampilkan data hakim.
13	Hapus Hakim	*Input*	Tombol untuk menghapus hakim.

5.1.5.11 Perancangan antarmuka halaman kejaksaan – *dashboard*

Halaman kejaksaan – dashboard ditampilkan awal ketika aktor telah melakukan autentikasi pada sistem dan memiliki role kejaksaan. Pada halaman ini ditampilkan statistik data tilang berdasarkan minggu dan barang bukti pelanggaran pada bentuk grafik untuk memudahkan pembacaan data. Gambar perancangan antarmuka ditunjukkan pada gambar 5.38 dan penjelasan dari perancangan antarmuka dijelaskan pada tabel 5.11.



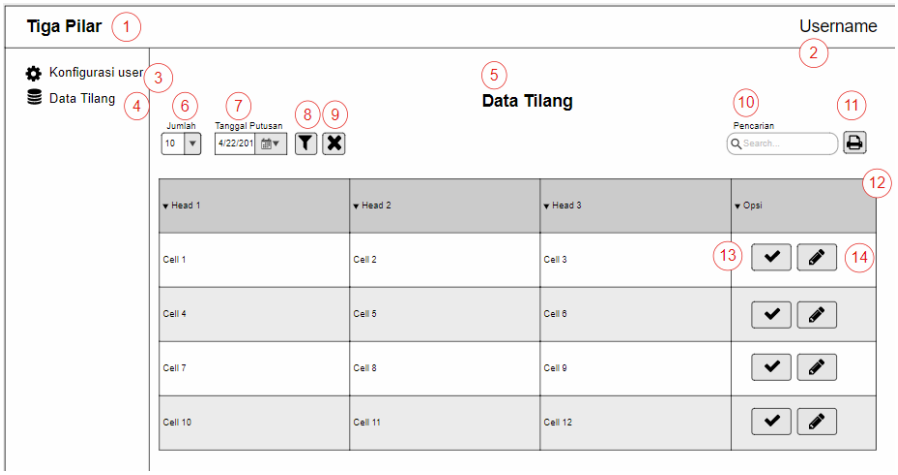
Gambar 5.38 Gambar perancangan antarmuka halaman kejaksaan – *dashboard*

Tabel 5.11 Penjelasan perancangan antarmuka halaman kejaksaan – *dashboard*

No	Nama	Tipe	Deskripsi
1	Tiga Pilar	Teks	Nama dari sistem.
2	Username	*Dropdown*	Nama dari *user* yang telah masuk.
3	Menu *dashboard*	*Link*	Menu untuk masuk ke halaman *dashboard*.
4	Menu data tilang	*Link*	Menu untuk masuk ke halaman data tilang.
5	*Header*	Teks	Judul dari halaman.
6	Judul chart	Teks	Nama dari *chart*
7	Data Tilang per Minggu	*Bar Chart*	*Chart* yang menampilkan jumlah data tilang per minggu berdasarkan tahun.
8	Judul Chart	Teks	Nama dari *chart.*
9	Data Tilang berdasarkan Barang Bukti	*Radar Chart*	*Chart* yang menampilkan jumlah data tilang berdasarkan barang bukti.

5.1.5.12 Perancangan antarmuka halaman kejaksaan – data tilang

Pada halaman kejaksaan – lihat data digunakan untuk melihat data berdasarkan minggu dan tahun, disini terdapat fitur untuk melakukan konfirmasi pembayaran, mengubah data jika diperlukan, dan fitur untuk mengunduh laporan dalam bentuk *file excel.* Gambar perancangan antarmuka ditunjukkan pada gambar 5.39 dan penjelasan dari perancangan antarmuka dijelaskan pada tabel 5.12.



Gambar 5.39 Gambar perancangan antarmuka halaman kejaksaan – data tilang

Tabel 5.12 Penjelasan perancangan antarmuka halaman kejaksaan – data tilang

No	Nama	Tipe	Deskripsi
1	Tiga Pilar	Teks	Nama dari sistem.
2	Username	*Dropdown*	Nama dari *user* yang telah masuk.
3	Menu *dashboard*	*Link*	Menu untuk masuk ke halaman *dashboard*.

4	Menu data tilang	*Link*	Menu untuk masuk ke halaman data tilang.
5	*Header*	Teks	Judul dari halaman.
6	Jumlah	*Dropdown*	*Dropdown* untuk menampilkan berapa jumlah data yang ditampilkan pada tabel.
7	Tahun	*Input*	*Input* untuk memilih tanggal putusan untuk melakukan *filtering* data.
8	*Filter*	Tombol	Tombol untuk melakukan *filtering* data.
9	*Clear*	Tombol	Tombol untuk menghapus *filtering* data dan menampilkan data pada minggu ini.
10	Pencarian	*Input*	*Input* teks untuk melakukan pencarian data
11	Unduh	Tombol	Tombol yang digunakan untuk mengunduh laporan tilang dalam bentuk *excel.*
12	Tabel data tilang	Tabel	Tabel untuk menampilkan data tilang.
13	Konfirmasi pembayaran	Tombol	Tombol digunakan untuk konfirmasi pembayaran
14	*edit*	Tombol	Tombol yang digunakan untuk mengubah data tilang.

- **Perancangan antarmuka iterasi kedua**

5.1.5.13 Perancangan antarmuka halaman kepolisian – barang bukti (iterasi kedua)

Pada halaman kepolisian – barang bukti digunakan untuk menampilkan data tilang dengan barang bukti yang belum diambil pada bulan tertentu. Gambar perancangan antarmuka ditunjukkan pada gambar 5.40 dan penjelasan dari perancangan antarmuka dijelaskan pada tabel 5.13.

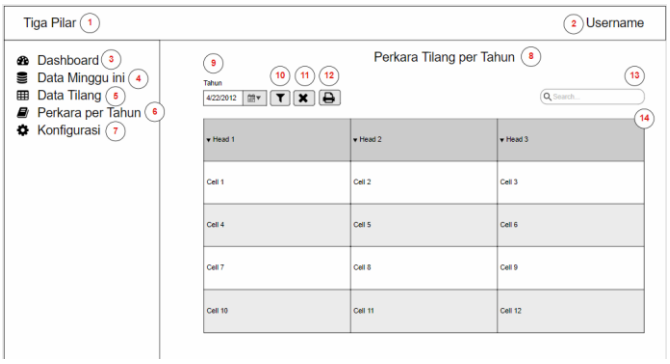
Gambar 5.40 Perancangan antarmuka halaman kepolisian – barang bukti (iterasi kedua)

Tabel 5.13 Penjelasan perancangan antarmuka halaman kepolisian – barang bukti (iterasi kedua)

No	Nama	Tipe	Deskripsi
1	Tiga Pilar	Teks	Nama dari sistem.
2	Username	*Dropdown*	Nama dari *user* yang telah masuk.
3	Menu *dashboard*	*Link*	Menu untuk masuk ke halaman *dashboard*.
4	Menu data tilang	*Link*	Menu untuk masuk ke halaman data tilang.
5	Menu *upload*	*Link*	Menu untuk masuk ke halaman upload data tilang.
6	Menu barang bukti	*Link*	Menu untuk masuk ke halaman barang bukti.
7	*Header*	Teks	Judul dari halaman.
8	Jumlah	*Dropdown*	*Dropdown* untuk menampilkan berapa jumlah data yang ditampilkan pada tabel.
9	Bulan	*Input*	*Input* untuk memilih bulan untuk melakukan *filtering* data.
10	*Filter*	Tombol	Tombol untuk melakukan *filtering* data.
11	*Clear*	Tombol	Tombol untuk menghapus *filtering* data dan menampilkan data saat ini.
12	Pencarian	*Input*	*Input* teks untuk melakukan pencarian data
13	Tabel data tilang	Tabel	Tabel untuk menampilkan data tilang.

5.1.5.14 Perancangan antarmuka halaman pengadilan – perkara per tahun (iterasi kedua)

Pada halaman pengadilan – perkara per tahun digunakan untuk melihat rekap perkara per tahun dan pada halaman ini juga terdapat fitur untuk mengunduh laporan perkara per tahun. Gambar perancangan antarmuka ditunjukkan pada gambar 5.41 dan penjelasan dari perancangan antarmuka dijelaskan pada tabel 5.14.



Gambar 5.41 Perancangan antarmuka halaman pengadilan – perkara per tahun (iterasi kedua)

Tabel 5.14 Penjelasan perancangan antarmuka halaman pengadilan – perkara per tahun (iterasi kedua)

No	Nama	Tipe	Deskripsi
1	Tiga Pilar	Teks	Nama dari sistem.
2	Username	*Dropdown*	Nama dari *user* yang telah masuk.
3	Menu *dashboard*	*Link*	Menu untuk masuk ke halaman *dashboard*.
4	Menu data minggu ini	*Link*	Menu untuk masuk ke halaman data minggu ini.
5	Menu data tailng	*Link*	Menu untuk masuk ke halaman data tilang.
6	Menu perkara per tahun	*Link*	Menu untuk masuk ke halaman perkara per tahun.
7	Menu konfigurasi	*Link*	Menu untuk masuk ke halaman perkara per tahun.
8	*Header*	Teks	Judul dari halaman.
9	Tahun	*Input*	*Input* untuk memilih tahun untuk melakukan *filtering* data.
10	*Filter*	Tombol	Tombol untuk melakukan *filtering* data.
11	*Clear*	Tombol	Tombol untuk menghapus *filtering* data dan menampilkan data saat ini.
12	*download*	Tombol	Tombol untuk mengunduh laporan perkara per tahun.
12	Pencarian	*Input*	*Input* teks untuk melakukan pencarian data
13	Tabel perkara	Tabel	Tabel untuk menampilkan data perkara per tahun.

5.2 Implementasi Sistem

Pada implementasi akan dijelaskan mengenai implementasi yang dilakukan dalam pembuatan sistem. Sebelum dijelaskan mengenai implementasi, akan dijelaskan mengenai spesifikasi yang digunakan dalam implementasi sistem. Spesifikasi tersebut berupa spesifikasi sistem yang digunakan, serta batasan dalam implementasi. Implementasi yang digunakan adalah implementasi basis data yang menggunakan physical data model, implementasi antarmuka, implementasi kelas-kelas yang digunakan atau implementasi kode program dalam sistem.

5.2.1 Spesifikasi Sistem

Dalam mengimplementasikan hasil dar analisis kebutuhan, perancangan, dan hasil dari evaluasi pengguna dibutuhkan perangkat keras serta perangkat lunak. Perangkat keras dan perangkat lunak perlu dspesifikasikan agar dapat mengetahui standar yang diperlukan dalam pembuatan sistem selama penelitian.

5.2.1.1 Spesifikasi perangkat keras

Dalam implementasi sistem informasi tiga pilar, dibutuhkan perangkat keras berupa laptop. Laptop digunakan untuk melakukan implementasi atau pembuatan aplikasi mulai dari implementasi antarmuka hingga implementasi kelas. Spesifikasi perangkat keras laptop yang digunakan dalam implementasi dapat dilihat pada tabel 5.15:

Tabel 5.15 Spesifikasi perangkat keras laptop

No	Nama Komponen	Spesifikasi
1	*Processor*	AMD FX-7600P
2	*Memory*	8 GB, 1 TB HDD
3	*Display*	Radeon R7

5.2.1.2 Spesifikasi perangkat lunak

Dalam implementasi sistem informasi tiga pilar, dibutuhkan perangkat lunak yang terdapat dalam laptop. Spesifikasi perangkat lunak laptop yang digunakan dalam implementasi dapat dilihat pada tabel 5.16:

Tabel 5.16 Spesifikasi perangkat lunak laptop

No	Nama Komponen	Spesifikasi
1	Sistem Operasi	Windows 10 Pro 64 bit
2	Bahasa Pemrograman	Java
3	Desain UI	Atom 1.21.1
4	IDE untuk pemrograman	STS 3.8.0.RELEASE
5	RDBMS	PostgreSQL dan PgAdmin III

5.2.1.3 Batasan implementasi

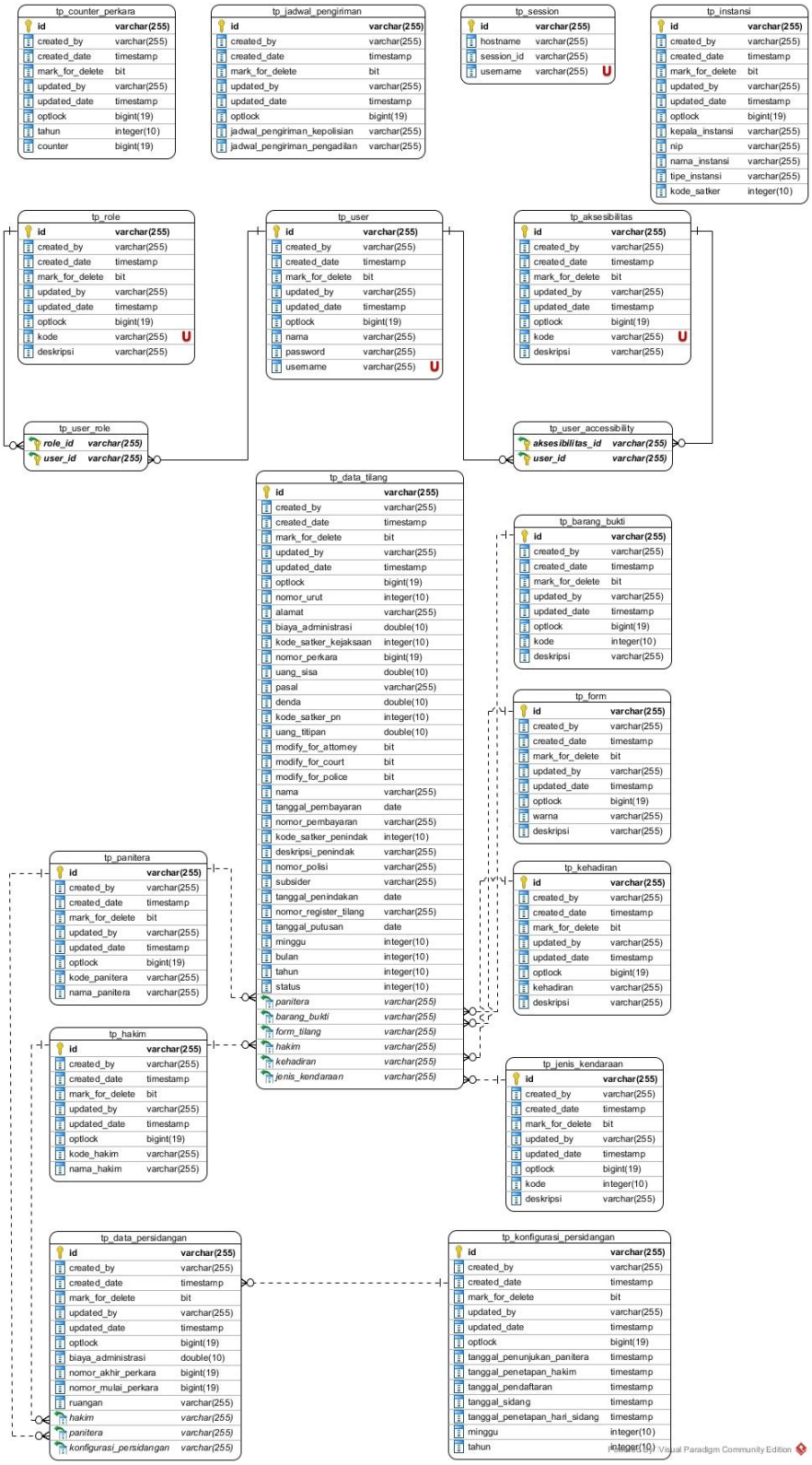
Dalam implementasi sistem informasi tiga pilar terdapat beberapa batasan sebagai berikut :

1. Sistem dikembangkan dengan metode *agile* dengan maksimal 3 kali iterasi.
2. Sistem dirancang sesuai kebutuhan dari tiga pilar di kota Kediri.
3. Sistem dirancang dapat berjalan dengan Java JDK versi 8.
4. Sistem harus terhubung dengan internet.

5.2.2 Implementasi Basis Data

Diagram *physical data model* (PDM) digunakan untuk menggambarkan tabel-tabel pada basis data relasional yang diimplementasikan pada sistem. PDM dibuat berdasarkan dari perancangan ERD sebelumnya. Pada PDM merepresentasikan bagaimana data disusun dan direlasikan pada DBMS, untuk itu sangat penting mengetahui konvensi dan batasan pada DBMS yang akan digunakan. Pada PDM sudah jelas mengenai tipe data, relasi antar entitas, *primary key, foreign key,*

unique key, ataupun batasan lain yang akan digunakan. Diagram PDM digambarkan pada gambar 5.42.



Gambar 5.42 Diagram *physical data model*

5.2.3 Implementasi Kode Program

Pada sub bab ini menjelaskan tentan implementasi kode program pada sistem informasi tiga pilar. Adapun implementasi kode program dibagi menjadi 2 yaitu implementasi kode program *web service* dan implementasi kode program AngularJs. Implementasi kode program *web service* digunakan sebagai *server* yang mengelola *request* dan *response* dari *client*, sedangkan implementasi kode program AngularJs digunakan sebagai penjembatan antara *web service* dan *client*. Adapun implementasi disini hanya dijelaskan beberapa fungsi utama dari sistem.

5.2.3.1 Implementasi kode program *web service*

Implementasi kode program *web service* merupakan implementasi dari kelas-kelas yang dirancang sebelumnya pada tahap perancangan. Pada implementasi ini akan dijelaskan implementasi *method* pada sistem yang penting dari *web service*.

Implementasi *service implementation*

Implementasi *service implementation* merupakan kelas-kelas yang digunakan untuk melakukan operasi *business logic* pada sistem. Adapun yang dijelaskan pada implementasi *layer service* adalah beberapa *method* utama dalam alur penyelesaian perkara tilang yang dianggap penting yang digunakan oleh sistem.

1. upload(file)

Method `upload(file)` digunakan untuk mengunggah *file excel* data tilang yang dapat dilihat pada tabel 5.17. Penjelasan *method* ini dapat dilihat pada tabel 5.18.

Tabel 5.17 Implementasi *method* upload(file) pada kelas TrafficTicketServiceImpl

No	***Method* upload(file) pada kelas TrafficTicketServiceImpl**
1	`@Override`
2	`@Transactional(readOnly = false, rollbackFor = Exception.class)` `public void upload(MultipartFile file) throws Exception {`
3	`ScheduleDto scheduleDto = scheduleService.findAll();`
4	`if (tigaPilarUtils.getCurrentLocaleDay()`
5	`.equalsIgnoreCase(scheduleDto.getPoliceDeliverySchedule()) &&` `!fileService.isFileExist()) {`
6	`int year = calendar.get(Calendar.YEAR);`
7	`int week = calendar.get(Calendar.WEEK_OF_YEAR);`
8	`String path = String.valueOf(year);`
9	`String name = "Data_Tilang_" + year + "_" + week;`
10	`String fileUploadPath = fileService.uploadFile(file, path,` `name);`
11	`excelUtil.insertDataFromExcelFile(fileUploadPath);`
12	`} else if (fileService.isFileExist()) {`
13	`throw new InvalidException("Anda telah mengirimkan berkas` `untuk minggu ini.");`
14	`} else {`
15	`throw new InvalidException("Bukan hari pengiriman data` `tilang.");`
16	`}`
17	`}`

Tabel 5.18 Penjelasan *method* upload(file) pada kelas TrafficTIcketServiceImpl

No	Penjelasan
1	Anotasi ini digunakan untuk melakukan *overriding* dari kelas *interface* TrafficTicketService.
2	Anotasi ini digunakan untuk mengakses ke basis data maka dibuat sebuah transaksi, dimana data akan disimpan pada basis data ketika semua transaksi berhasil, jika tidak maka transaksi akan dianggap gagal.
3	Inisialisasi nama *method.*
4	Memanggil *method* findAll() pada kelas scheduleService untuk mendapatkan jadwal pengiriman yang akan disimpan pada variabel scheduleDto.
5	Jika nilai dari pemanggilan *method* getCurrentLocalDay() pada kelas TigaPilarUtils sama dengan nilai dari *method* getPoliceDeliverySchedule() pada variabel scheduleDto dan nilai dari pemanggilan *method* isFileExist() dari kelas FileService bernilai false maka akan dijalankan operasi didalamnya.
6	Inisialisasi variabel year dengan nilai tahun saat ini.
7	Inisialisasi variabel week dengan nilai minggu saat ini.
8	Inisialisasi variabel path dengan nilai konversi variabel year dijadikan string.
9	Inisialiasi variabel name dengan nilai Data_Tilang_(nilai variabel year)_(nilai variabel week).
10	Inisialisasi variabel fileUploadPath dengan nilai dari *method* uploadFile(file, path, name).
11	Memanggil *method* insertDataFromFileExcel(fileUploadPath) dari kelas excelUtil.
12-13	Jika nilai dari pemanggilan *method* isFileExist() dari kelas FileService bernilai false maka akan dilempar InvalidException.
14-15	Jika tidak semuanya maka akan dilempar InvalidException.

2. sendDataToCourt()

Method `sendDataToCourt()` digunakan untuk mengirimkan data tilang yang telah diunggah dari kepolisian ke pengadilan yang dapat dilihat pada tabel 5.19. penjelasan *method* ini dapat dilihat pada tabel 5.20.

Tabel 5.19 Implementasi *method* sendDataToCourt() pada kelas TrafficTicketServiceImpl

No	*Method* sendDataToCourt() pada kelas TrafficTicketServiceImpl
1	`@Override`
2	`@Transactional(readOnly = false, rollbackFor = Exception.class)`
3	`public void sendDataToCourt() throws Exception {`
4	`CounterDto counterDto = counterService.findByYear();`
5	`Institution institution = institutionRepository.findByInstitutionType(InstitutionType.PENGADILAN);`
6	`List<TrafficTicket> trafficTickets = trafficTicketRepository.findWeeklyForPolice();`
7	`long counter = counterDto.getCounter();`
8	`for (TrafficTicket trafficTicket : trafficTickets) {`

```
9           trafficTicket.setCaseNumber(++counter);
10          trafficTicket.setModifyForPolice(false);
11          trafficTicket.setCourtCode(institution.getInstitutionWorkUn
12          itCode());
13      }

14      counterDto.setCounter(counter);
15      trafficTicketRepository.save(trafficTickets);
16      counterService.save(counterDto);
17  }
```

Tabel 5.20 Penjelasan *method* sendDataToCourt() pada kelas TrafficTicketServiceImpl

No	Penjelasan
1	Anotasi ini digunakan untuk melakukan *overriding* dari kelas *interface* TrafficTicketService.
2	Anotasi ini digunakan untuk mengakses ke basis data maka dibuat sebuah transaksi, dimana data akan disimpan pada basis data ketika semua transaksi berhasil, jika tidak maka transaksi akan dianggap gagal.
3	Inisialisasi nama *method.*
4	Inisialisasi variabel counterDto dengan nilai dari pemanggilan *method* findByYear() dari kelas counterService.
5	Inisialisasi variabel institution dengan nilai dari pemanggilan *method* findByInstitutionType(institutionType) dari kelas InstitutionRepository.
6	Inisialisasi variabel trafficTickets dengan nilai dari pemanggilan *method* findWeeklyForPolice() dari kelas TrafficTicketRepository.
7	Inisialisasi variabel counter dengan nilai dari pemanggila *method* getCounter() dari variabel counterDto.
8-13	Melakukan *looping* sebanyak panjang trafficTickets dan mengeset counter dengan nilai ++counter, mengeset nilai modifyForPolice dengan nilai false, dan mengeset courtCode dengan nilai dari pemanggilan *method* getInstitutionWorkUnitCode() dari variabel institution.
14	Mengeset nilai counter pada variabel counterDto dngan nilai dari variabel counter.
15	Menyimpan trafficTickets ke basis data
16	Menyimpan counterDto melalui *method* save(counterDto) dari kelas counterService.

3. save(trialConfiurationDto)

Method `save(trialConfigurationDto)` digunakan untuk menyimpan data konfigurasi persidangan yang dimasukkan oleh pengadilan, implementasi *method* ini dapat dilihat pada tabel 5.21 dan penjelasan dari implementasi ini dapat dilihat pada tabel 5.22.

Tabel 5.21 Implementasi *method* save(trialConfigurationDto) pada kelas TrialConfigurationServiceImpl

No	*Method* save(trialConfigurationDto) pada kelas TrialConfigurationServiceImpl
1 2	`@Override` `@Transactional(readOnly = false, rollbackFor = Exception.class)`

```
3   public void save(TrialConfigurationDto trialConfigurationDto)
    throws Exception {

4       TrialConfiguration trialConfiguration =
        trialConfigurationRepository.findByYearAndWeek(trialConfigurat
        ionDto.getYear(), calendar.get(Calendar.WEEK_OF_YEAR));
5       trialConfigurationDto.setWeek(calendar.get(Calendar.WEEK_OF_YE
        AR));

6      if (trialConfiguration == null) {
7         trialConfiguration = trialConfigurationRepository
          .save(TrialConfigurationConverter.toEntity(trialConfigurati
          onDto));

8         for (TrialDataDto trialDataDto:
    trialConfigurationDto.getTrialDatas()) {

9            TrialData trialData =
             TrialDataConverter.toEntity(trialDataDto);
10           Judge judge =
             judgeRepository.findByCode(trialDataDto.getJudge());
11           Clerks clerks =
             clerksRepository.findByCode(trialDataDto.getClerks());
12           Presence presence =
             presenceRepository.findByPresence("Verstek");
13          trialData.setJudge(judge);
14          trialData.setClerks(clerks);
15          trialData.setTrialConfiguration(trialConfiguration);

16           trialDataRepository.save(trialData);
17           trafficTicketRepository.updateTicketForCourt(trialConfigu
             rationDto.getTrialDate(), judge, clerks,
             trialDataDto.getAdministrativeCost(), presence, "3 (tiga)
             hari kurungan", trialDataDto.getCaseNumberStart(),
             trialDataDto.getCaseNumberEnd(),trialConfigurationDto.get
             Year(), trialConfigurationDto.getWeek());
18        }
19     } else {
20           throw new EntityExistsException("Konfigurasi Persidangan
             untuk tahun: " + calendar.get(Calendar.YEAR) + " dan
             minggu: " + calendar.get(Calendar.WEEK_OF_YEAR) + " sudah
             ada.");
21     }
22   }
```

Tabel 5.22 Penjelasan *method* save(trialConfigurationDto) pada kelas TrialConfigurationServiceImpl

No	Penjelasan
1	Anotasi ini digunakan untuk melakukan *overriding* dari kelas *interface* TrafficTicketService.
2	Anotasi ini digunakan untuk mengakses ke basis data maka dibuat sebuah transaksi, dimana data akan disimpan pada basis data ketika semua transaksi berhasil, jika tidak maka transaksi akan dianggap gagal.
3	Inisialisasi nama *method.*
4	Inisialisasi variabel trialConfiguration dengan nilai dari pemanggilan *method* findByYearAndWeek(year, week) dari kelas trialConfigurationRepository.
5	Mengeset nilai week pada variabel trialConfigurationDto dengan nilai minggu ini.
6	Jika nilai trialConfiguration sama dengan null maka akan dijalankan proses didalamnya.

7	Inisialisasi variabel trialConfiguration dengan nilai dari pemanggilan *method* save(trialConfiguration) dari kelas trialConfigurationRepository.
8	Melakukan *looping* sebanyak panjang trialData dari variabel trialConfigurationDto dan menjalankan proses didalamnya.
9	Inisialisasi variabel trialData dengan nilai dari konversi trialDataDto.
10	Inisialisasi variabel judge dengan nilai dari pemanggilan *method* findByCode(code) dari kelas JudgeRepository.
11	Inisialisasi variabel clerks dengan nilai dari pemanggilan *method* findByCode(code) dari kelas clerksRepository.
12	Inisialisasi variabel presence dengan nilai dari pemanggilan *method* findByPresence(code) dari kelas PresenceRepository.
13	Mengeset nilai judge pada variabel trialData dengan variabel judge.
14	Mengeset nilai clerks pada variabel trialData dengan variabel clerks.
15	Mengeset nilai trialConfiguratino pada variabel trialData dengan variabel trialConfiguration.
16	Menyimpan trialData ke basis data.
17	Memperbarui nilai data tilang pada basis data.
19-21	Jika nilai trialConfiguration lainnya maka akan melempar EntityExistException.

4. inputCost(ticketRegisterNumber, cost)

Method `inputCost(ticketRegisterNumber, cost)` digunakan untuk memasukkan denda pada data tilang, implementasi *method* ini dapat dilihat pada tabel 5.23, dan penjelasan dari implementasi *method* ini dapat dilihat pada tabel 5.24.

Tabel 5.23 Implementasi *method* inputCost(ticketRegisterNumber, cost) pada kelas TrafficTicketServiceImpl

No	*Method* inputCost(ticketRegisterNumber, cost) pada kelas TrafficTicketServiceImpl
1	`@Override`
2	`@Transactional(readOnly = false, rollbackFor = Exception.class)`
3	`public void inputCost(String ticketRegisterNumber, double cost) throws Exception {`
4	`TrafficTicket trafficTicket = trafficTicketRepository .findByTicketRegisterNumberAndYear(ticketRegisterNumber, calendar.get(Calendar.YEAR));`
5	`if (trafficTicket != null) {`
6	`if (trafficTicket.getPaymentDate() != null) {`
7	`double changeMoney = trafficTicket.getDepositMoney() - (cost + trafficTicket.getAdministrativeCost());`
8	`trafficTicket.setCost(cost);`
9	`trafficTicket.setChangeMoney(changeMoney);`
10	`} else {`
11	`trafficTicket.setCost(cost);`
12	`}`
13	`trafficTicketRepository.save(trafficTicket);`
14	`} else {`
15	

16 17	throw new EntityNotFoundException("Data tilang tidak ditemukan."); } }

Tabel 5.24 Penjelasan *method* inputCost(ticketRegisterNumber, cost) pada kelas TrafficTicketServiceImpl

No	Penjelasan
1	Anotasi ini digunakan untuk melakukan *overriding* dari kelas *interface* TrafficTicketService.
2	Anotasi ini digunakan untuk mengakses ke basis data maka dibuat sebuah transaksi, dimana data akan disimpan pada basis data ketika semua transaksi berhasil, jika tidak maka transaksi akan dianggap gagal.
3	Inisialisasi nama *method.*
4	Inisialisasi variabel trafficTicket dengan nilai dari pemanggilan *method* findByTicketRegisterNumberAndYear(ticketRegisterNumber, year) dari kelas TrafficTicketRepository.
5	Jika nilai trafficTicket tidak sama dengan null maka dijalankan proses didalamnya.
6	Jika nilai paymentDate dari variabel trafficTicket tidak sama dengan null maka dijalankan proses didalamnya.
7	Inisialisasi variabel changeMoney dengan nilai dari kalkulasi depositMoney – (cost + administrativeCost).
8	Mengeset nilai cost pada variabel trafficTicket dengan nilai dari variabel cost.
9	Mengeset nilai changeMoney pada variabel trafficTicket dengan nilai dari variabel changeMoney.
10-12	Jika nilai paymentDate dari variabel trafficTIcket selain tidak sama dengan null maka mengeset nilai cost dari variabel trafficTicket dengan nilai dari variabel cost.
13	Menyimpan trafficTicket kedalam basis data.
14-16	Jika nilai dari variabel trafficTicket selain tidak sama dengan null maka akan melempar EntityNotFoundException.

5. inputPayment(year, ticketRegisterNumber, paymentDto)

Method `inputPayment(year, ticketRegisterNumber, paymentDto)` digunakan untuk memasukkan konfirmasi pembayaran tilang, implementasi *method* ini dapat dilihat pada tabel 5.25, dan penjelasan dari *method* ini dapat dilhat pada tabel 5.26.

Tabel 5.25 Implementasi *method* inputPayment(year, ticketRegisterNumber, paymentDto) pada kelas TrafficTicketServiceImpl

No	*Method* inputPayment(year, ticketRegisterNumber, paymentDto) pada kelas TrafficTicketServiceImpl
1 2 3	@Override @Transactional(readOnly = false, rollbackFor = Exception.class)

```
    public void inputPayment(int year, String ticketRegisterNumber,
4   PaymentDto paymentDto) throws Exception {
        TrafficTicket trafficTicket =
        trafficTicketRepository.findByTicketRegisterNumberAndYear(tick
5       etRegisterNumber, year);
6       if (trafficTicket != null) {
7         if(paymentDto.getPaymentType().equalsIgnoreCase("bank")) {
8           trafficTicket.setPaymentDate(paymentDto.getPaymentDate());
9           trafficTicket.setStatus(1);
          } else if (
10        paymentDto.getPaymentType().equalsIgnoreCase("tempat")) {
11            trafficTicket.setPaymentDate(new Date());
12            trafficTicket.setStatus(2);
          } else if (
13        paymentDto.getPaymentType().equalsIgnoreCase("kejaksaan")){
14            trafficTicket.setStatus(3);
15      }
16      trafficTicketRepository.save(trafficTicket);
17    } else {
          throw new EntityNotFoundException("Data tilang tidak
18        ditemukan.");
19    }
    }
```

Tabel 5.26 Penjelasan *method* inputPayment(year, ticketRegisterNumber, paymentDto) pada kelas TrafficTicketServiceImpl

No	Penjelasan
1	Anotasi ini digunakan untuk melakukan *overriding* dari kelas *interface* TrafficTicketService.
2	Anotasi ini digunakan untuk mengakses ke basis data maka dibuat sebuah transaksi, dimana data akan disimpan pada basis data ketika semua transaksi berhasil, jika tidak maka transaksi akan dianggap gagal.
3	Inisialisasi nama *method.*
4	Inisialisasi variabel trafficTicket dengan nilai dari pemanggilan *method* findByTicketRegisterNumberAndYear(ticketRegisterNumber, year) dari kelas TrafficTicketRepository.
5	Jika nilai trafficTicket tidak sama dengan null maka dijalankan proses didalamnya.
6	Jika nilai paymentType dari variabel paymentDto sama dengan bank maka dijalankan proses didalamnya.
7	Mengeset nilai paymentDate dari variabel trafficTicket dengan nilai dari paymentDate dari variabel paymentDto.
8	Mengeset nilai status dari variabel trafficTicket dengan nilai 1.
9	Jika nilai paymentType dari variabel paymentDto sama dengan tempat maka dijalankan proses didalamnya.
10	Mengeset nilai paymentDate dari variabel trafficTicket dengan nilai dari tanggal hari ini.
11	Mengeset nilai status dari variabel trafficTicket dengan nilai 2.
12	Jika nilai paymentType dari variabel paymentDto sama dengan kejaksaan maka dijalankan proses didalamnya.

13	Mengeset nilai status dari variabel trafficTicket dengan nilai 3.
15	Menyimpan trafficTicket kedalam basis data.
16-18	Jika nilai dari trafficTicket selain tidak sma dengan null maka akan dilempar EntityNotFoundException.

6. findAvailableEvidence(year, month)

Method `findAvailableEvidence(year, month)` diimplementasikan pad iterasi kedua, *method* ini digunakan untuk menampilkan data tilang dengan barang bukti yang belum diambil atau masih dikejaksaan per bulannya, implementasi dari *method* ini dapat dilihat pada tabel 5.27, dan penjelasan dari *method* ini dapat dilihat pada tabel 5.28.

Tabel 5.27 Implementasi *method* findAvailableEvidence(year, month) pada kelas TrafficTicketServiceImpl (iterasi kedua)

No	*Method* findAvailableEvidence(year, month) pada kelas TrafficTicketServiceImpl iterasi kedua
1	`@Override`
2	`public List<TrafficTicket> findAvailableEvidence(int year, int month) throws Exception {`
3	`List<TrafficTicket> trafficTickets = new ArrayList<>();`
4	`List<Integer> status = new ArrayList<>();`
5	`status.add(0);`
6	`status.add(3);`
7	`if (year == 0 && month == 0) {`
8	`trafficTickets = trafficTicketRepository.findByYearAndMonthAndStatusIn(calendar.get(Calendar.YEAR), calendar.get(Calendar.MONTH) + 1, status);`
9	`} else {`
10	`trafficTickets = trafficTicketRepository.findByYearAndMonthAndStatusIn(year, month, status);`
11	`}`
12	`return trafficTickets;`
13	`}`

Tabel 5.28 Penjelasan *method* findAvailableEvidence(year, month) pada kelas TrafficTicketServiceImpl (iterasi kedua)

No	Penjelasan
1	Anotasi ini digunakan untuk melakukan *overriding* dari kelas *interface* TrafficTicketService.
2	Inisialisasi nama *method.*
3	Inisialisasi variabel trafficTickets.
4	Inisialisasi variabel status.
5	Menambahkan nilai 0 ke variabel status.
6	Menambahkan nilai 3 ke variabel status.
7	Jika nilai variabel year dan month sama dengan 0 maka dijalankan proses didalamnya.

8	Mengeset nilai variabel trafficTickets dengan nilai dari pemanggilan *method* findByYearAndMonthAndStatusIn(year, month, status) dari kelas TrafficTicketRepository.
9	Jika nilai variabel year dan month selain sama dengan 0 maka dijalankan proses didalamnya.
10	Mengeset nilai variabel trafficTickets dengan nilai dari pemanggilan *method* findByYearAndMonthAndStatusIn(year, month, status) dari kelas TrafficTicketRepository.
11	Mengembalikan nilai dari trafficTickets.

Implementasi *controller*

Implementasi *controller* merupakan kelas-kelas yang digunakan sebagai RESTful *web service*. Adapun yang dijelaskan pada implementasi *controller* adalah beberapa *method* utama dalam alur penyelesaian perkara tilang yang dianggap penting yang digunakan oleh sistem.

1. upload(requestId, file)

Method `upload(requestId, file)` adalah *method* yang akan di*mapping* menjadi *web service* dengan URI /TigaPilar/api/ticket/upload, implementasi dari *method* ini dapat dilihat pada tabel 5.29, dan penjelasan dari *method* ini akan dijelaskan pada tabel 5.30.

Tabel 5.29 Implementasi *method* upload(requestId, file) pada kelas TrafficTicketController

No	***Method* upload(requestId, file) pada kelas TrafficTicketController**
1	`@RequestMapping(value = UPLOAD, method = RequestMethod.POST, consumes = {MediaType.MULTIPART_FORM_DATA_VALUE}, produces = {MediaType.APPLICATION_JSON_VALUE})`
2	`public BaseResponse upload(@RequestParam String requestId, @RequestPart MultipartFile file) throws Exception {`
3	`if (Credential.getRole().contains(RoleEnum.KEPOLISIAN)&& Credential.getAccessibility().contains(AccessibilityEnum.CREATE )) {`
4	`trafficTicketService.upload(file);`
5	`return new BaseResponse(true, requestId, HttpStatus.OK, "Data tilang berhasil disimpan");`
6	`} else`
7	`throw new UnauthorizedException(UNAUTHORIZED_MESSAGE);`
8	`}`

Tabel 5.30 Penjelasan *method* upload(requestId, file) pada kelas TrafficTicketController

No	Penjelasan
1	Anotasi ini digunakan untuk mengubah method menjadi API *path* dari *web service*, anotasi ini memiliki parameter value yang merupakan API *path* / URI, method yang merupakan HTTP *method* yang digunakan dari *web service*, dan consumes yang merupakan jenis data yang dapat diterima oleh *web service* ini, dan produces yang merupakan jenis data yang dikembalikan oleh *web service* ini.
2	Inisialisasi nama *method.*

3	Jika nilai role dari variabel Credential sama dengan nilai RoleEnum.KEPOLISIAN dan nilai accessibility dari variabel Credential sama dengan nilai AccessibilityEnum.CREATE maka dijalankan proses didalamnya.
4	Memanggil *method* upload(file) dari kelas TrafficTIcketService.
5	Mengembalikan *response.*
6-8	Jika nilai role dan accessibility selain itu maka melempar UnauthorizedException.

2. sendDataToCourt(requestId)

Method `sendDataToCourt(requestId)` adalah *method* yang akan di*mapping* menjadi *web service* dengan URI /TigaPilar/api/ticket/send, implementasi dari *method* ini dapat dilihat pada tabel 5.31, dan penjelasan dari *method* ini akan dijelaskan pada tabel 5.32.

Tabel 5.31 Implementasi *method* sendDataToCourt(requestId) pada kelas TrafficTicketController

No	***Method* sendDataToCourt(requestId) pada kelas TrafficTicketController**
1	`@RequestMapping(value = SEND, method = RequestMethod.PUT, produces = {MediaType.APPLICATION_JSON_VALUE})`
2	`public BaseResponse sendDataToCourt(@RequestParam String requestId) throws Exception {`
3	`if (Credential.getRole().contains(RoleEnum.KEPOLISIAN) && Credential.getAccessibility().contains(AccessibilityEnum.UPDATE )) {`
4	`trafficTicketService.sendDataToCourt();`
5	`return new BaseResponse(true, requestId, HttpStatus.OK, "Data berhasil dikirimkan ke pengadilan");`
6	`} else`
7	`throw new UnauthorizedException(UNAUTHORIZED_MESSAGE);`
8	`}`

Tabel 5.32 Penjelasan *method* sendDataToCourt(requestId) pada kelas TrafficTicketController

No	Penjelasan
1	Anotasi ini digunakan untuk mengubah method menjadi API *path* dari *web service*, anotasi ini memiliki parameter value yang merupakan API *path* / URI, method yang merupakan HTTP *method* yang digunakan dari *web service*, dan produces yang merupakan jenis data yang dikembalikan oleh *web service* ini.
2	Inisialisasi nama *method.*
3	Jika nilai role dari variabel Credential sama dengan nilai RoleEnum.KEPOLISIAN dan nilai accessibility dari variabel Credential sama dengan nilai AccessibilityEnum.UPDATE maka dijalankan proses didalamnya.
4	Memanggil *method* sendDataToCourt() dari kelas TrafficTIcketService.
5	Mengembalikan *response.*
6-8	Jika nilai role dan accessibility selain itu maka melempar UnauthorizedException.

3. save(requestId, trialConfigurationDto)

Method `save(requestId, trialConfigurationDto)` adalah *method* yang akan di*mapping* menjadi *web service* dengan URI /TigaPilar/api/trial, implementasi dari *method* ini dapat dilihat pada tabel 5.33, dan penjelasan dari *method* ini akan dijelaskan pada tabel 5.34.

Tabel 5.33 Implementasi *method* save(requestId, trialConfigurationDto) pada kelas TrialConfigurationController

No	*Method* save(requestId, trialConfigurationDto) pada kelas TrialConfigurationController
1	`@RequestMapping(method = RequestMethod.POST, consumes = {MediaType.APPLICATION_JSON_VALUE},produces = {MediaType.APPLICATION_JSON_VALUE})`
2	`public BaseResponse save(@RequestParam String requestId, @RequestBody TrialConfigurationDto trialConfigurationDto) throws Exception {`
3	`if (Credential.getRole().contains(RoleEnum.PENGADILAN) && Credential.getAccessibility().contains(AccessibilityEnum.CREATE )) {`
4	`trialConfigurationService.save(trialConfigurationDto);`
5	`return new BaseResponse(true, requestId, HttpStatus.OK, "Konfigurasi persidangan berhasil disimpan");`
6	`} else {`
7	`throw new UnauthorizedException(UNAUTHORIZED_MESSAGE);`
8	`}`
9	`}`

Tabel 5.34 Penjelasan *method* save(requestId, trialConfigurationDto) pada kelas TrialConfigurationController

No	Penjelasan
1	Anotasi ini digunakan untuk mengubah method menjadi API *path* dari *web service*, anotasi ini memiliki parameter value yang merupakan API *path* / URI, method yang merupakan HTTP *method* yang digunakan dari *web service*, consumes yang merupakan jenis data yang dapat diterima *web service* ini, dan produces yang merupakan jenis data yang dikembalikan oleh *web service* ini.
2	Inisialisasi nama *method.*
3	Jika nilai role dari variabel Credential sama dengan nilai RoleEnum.PENGADILAN dan nilai accessibility dari variabel Credential sama dengan nilai AccessibilityEnum.CREATE maka dijalankan proses didalamnya.
4	Memanggil *method* save(trialConfigurationDto) dari kelas TrialConfigurationService.
5	Mengembalikan *response.*
6-8	Jika nilai role dan accessibility selain itu maka melempar UnauthorizedException.

4. inputCost(requestId, ticketRegisterNumber, cost)

Method `inputCost(requestId, ticketRegisterNumber, cost)` adalah *method* yang akan di*mapping* menjadi *web service* dengan URI /TigaPilar/api/ticket/input-cost, implementasi dari *method* ini dapat dilihat pada tabel 5.35, dan penjelasan dari *method* ini akan dijelaskan pada tabel 5.36.

Tabel 5.35 Implementasi *method* inputCost(requestId, ticketRegisterNumber, cost) pada kelas TrafficTicketController

No	***Method*** **inputCost(requestId, ticketRegisterNumber, cost) pada kelas TrafficTicketController**
1	`@RequestMapping(value = INPUT_COST, method = RequestMethod.POST, consumes = {MediaType.APPLICATION_JSON_VALUE})`
2	`public BaseResponse inputCost(@RequestParam String requestId,@RequestParam String ticketRegisterNumber, @RequestParam double cost) throws Exception {`
3	`if (Credential.getRole().contains(RoleEnum.PENGADILAN) && Credential.getAccessibility().contains(AccessibilityEnum.CREATE )) {`
4	`trafficTicketService.inputCost(ticketRegisterNumber, cost);`
5	`return new BaseResponse(true, requestId, HttpStatus.OK, "Berhasil menginputkan denda");`
6	`} else {`
7	`throw new UnauthorizedException(UNAUTHORIZED_MESSAGE);`
8	`}`
9	`}`

Tabel 5.36 Penjelasan *method* inputCost(requestId, ticketRegisterNumber, cost) pada kelas TrafficTicketController

No	Penjelasan
1	Anotasi ini digunakan untuk mengubah method menjadi API *path* dari *web service*, anotasi ini memiliki parameter value yang merupakan API *path* / URI, method yang merupakan HTTP *method* yang digunakan dari *web service*, dan produces yang merupakan jenis data yang dikembalikan oleh *web service* ini.
2	Inisialisasi nama *method.*
3	Jika nilai role dari variabel Credential sama dengan nilai RoleEnum.PENGADILAN dan nilai accessibility dari variabel Credential sama dengan nilai AccessibilityEnum.CREATE maka dijalankan proses didalamnya.
4	Memanggil *method* inputCost(ticketRegisterNumber, cost) dari kelas trafficTIcketService.
5	Mengembalikan *response.*
6-8	Jika nilai role dan accessibility selain itu maka melempar UnauthorizedException.

5. inputPayment(requestId, year, ticketRegisterNumber, paymentDto)

Method `inputPayment(requestId, year, ticketRegisterNumber, paymentDto)` adalah *method* yang akan di*mapping* menjadi *web service* dengan URI /TigaPilar/api/ticket/{year}/{ticketRegisterNumber, implementasi dari *method* ini dapat dilihat pada tabel 5.37, dan penjelasan dari *method* ini akan dijelaskan pada tabel 5.38.

Tabel 5.37 Implementasi *method* inputPayment(requestId, year, ticketRegisterNumber, paymentDto) pada kelas TrafficTicketController

No	***Method*** **inputPayment(requestId, year, ticketRegisterNumber, paymentDto) pada kelas TrafficTicketController**
1	`@RequestMapping(value = "/{year}" + "/{ticketRegisterNumber}", method = RequestMethod.PUT, consumes = {MediaType.APPLICATION_JSON_VALUE})`
2	

```
    public BaseResponse inputPayment(@RequestParam String requestId,
    @PathVariable int year, @PathVariable String ticketRegisterNumber,
    @RequestBody PaymentDto paymentDto)
3   throws Exception {
       if (Credential.getRole().contains(RoleEnum.KEJAKSAAN) &&
       Credential.getAccessibility().contains(AccessibilityEnum.UPDATE
4      )) {
           trafficTicketService.inputPayment(year,ticketRegisterNumber
5          , paymentDto);
           return new BaseResponse(true, requestId, HttpStatus.OK,
6          "Berhasil memasukkan tanggal pembayaran");
7      } else
8          throw new UnauthorizedException(UNAUTHORIZED_MESSAGE);
    }
```

Tabel 5.38 Penjelasan *method* inputPayment(requestId, year, ticketRegisterNumber, paymentDto) pada kelas TrafficTicketController

No	Penjelasan
1	Anotasi ini digunakan untuk mengubah method menjadi API *path* dari *web service*, anotasi ini memiliki parameter value yang merupakan API *path* / URI, method yang merupakan HTTP *method* yang digunakan dari *web service*, dan consumes yang merupakan jenis data yang dapat diterima oleh *web service* ini.
2	Inisialisasi nama *method.*
3	Jika nilai role dari variabel Credential sama dengan nilai RoleEnum.KEJAKSAAN dan nilai accessibility dari variabel Credential sama dengan nilai AccessibilityEnum.UPDATE maka dijalankan proses didalamnya.
4	Memanggil *method* inputPayment(year, ticketRegisterNumber, paymentDto) dari kelas trafficTIcketService.
5	Mengembalikan *response.*
6-8	Jika nilai role dan accessibility selain itu maka melempar UnauthorizedException.

6. findAvailableEvidence(requestId, year, month)

Method `findAvailableEvidence(requestId, year, month)` adalah *method* yang diimplementasikan pada iterasi kedua yang akan di*mapping* menjadi *web service* dengan URI /TigaPilar/api/ticket/available-evidence, implementasi dari *method* ini dapat dilihat pada tabel 5.39, dan penjelasan dari *method* ini akan dijelaskan pada tabel 5.40.

Tabel 5.39 Implementasi *method* findAvailableEvidence(requestId, year, month) pada kelas TrafficTicketController (iterasi kedua)

No	*Method* findAvailableEvidence(requestId, year, month) pada kelas TrafficTicketController
1	`@RequestMapping(value = AVAILABLE_EVIDENCE, method = RequestMethod.GET, produces = {MediaType.APPLICATION_JSON_VALUE})`
2	`public ListResponse<TrafficTicket> findAvailableEvidence(@RequestParam String requestId, @RequestParam(defaultValue = "0") int year, @RequestParam(defaultValue = "0") int month)throws Exception {`
3	`if (Credential.getRole().contains(RoleEnum.KEPOLISIAN) && Credential.getAccessibility().contains(AccessibilityEnum.READ )) {`

```
4           return new ListResponse<>(true, requestId, HttpStatus.OK,
            "Data Tilang dengan Barang Bukti yang belum diambil",
            trafficTicketService.findAvailableEvidence(year, month));
5       } else
6           throw new UnauthorizedException(UNAUTHORIZED_MESSAGE);
7   }
```

Tabel 5.40 Penjelasan *method* findAvailableEvidence(requestId, year, month) pada kelas TrafficTicketController (iterasi kedua)

No	Penjelasan
1	Anotasi ini digunakan untuk mengubah method menjadi API *path* dari *web service*, anotasi ini memiliki parameter value yang merupakan API *path* / URI, method yang merupakan HTTP *method* yang digunakan dari *web service*, dan produces yang merupakan jenis data yang dapat dikembalikan oleh *web service* ini.
2	Inisialisasi nama *method.*
3	Jika nilai role dari variabel Credential sama dengan nilai RoleEnum.KEPOLISIAN dan nilai accessibility dari variabel Credential sama dengan nilai AccessibilityEnum.READ maka dijalankan proses didalamnya.
4	Mengembalikan *response* dari *method* findAvailableEvidence(year, month) dari kelas TrafficTicketService.
5-7	Jika nilai role dan accessibility selain itu maka melempar UnauthorizedException.

5.2.3.2 Implementasi kode program *AngularJs*

Implementasi kode program *AngularJs* digunakan untuk menjembatani *client* untuk mengakses ke *web service* yang telah dibuat. Adapun pada implementasi *AngularJs* dibuat berdasarkan per halaman antarmuka yang telah dibuat sebelumnya. Pada implementasi kode program AngularJs yang dibahas adalah *module service* AngularJs yang berkomunikasi dengan *web service* yang telah diimplementasikan sebelumnya, fungsi lainnya tidak akan dibahas. *Module service* digunakan untuk mengakses API *path* / URI dari *web service* yang telah dibuat sebelumnya.

1. Implementasi *method* UploadFactory.post(file) pada UploadService

Method `UploadFactory.post(file)` digunakan untuk *mengakses web service* dengan URI /TigaPilar/api/ticket/upload yang telah dibuat sebelumnya. Implementasi *method* ini dapat dilihat pada tabel 5.41, penjelasan *method* ini dapat dilihat pada tabel 5.42.

Tabel 5.41 Implementasi *method* UploadFactory.post(file) pada UploadService

No	***Method* UploadFactory.post(file) pada *module* UploadService**

1
2
3
4
5
6
7
	`angular.module('uploadService').factory('UploadFactory',`
	`  function($http, RequestIdFactory) {`
	`  return {`
	`  post: function(file) {`
	`  var formData = new FormData();`
	`  formData.append('file', file);`
	`  return $http.post(Context +`
	`  '/api/ticket/upload?requestId=' +`
	`  RequestIdFactory.get(), formData, {`

```
8             transformRequest: angular.identity,
9             headers: {
10              'Content-Type': undefined
11            }
12          }).then(function(response) {
13            return response.data;
14          }, function(response) {
15            return response;
16          });
17        }
18      }
19    });
```

Tabel 5.42 Penjelasan *method* UploadFactory.post(file) pada UploadService

No	Penjelasan
1	Deklarasi nama *module* uploadService, dan nama *factory* UploadFactory.
2	Deklarasi pembuatan fungsi dengan parameter $http dan RequestIdFactory.
3	Pengembalian nilai dari *factory* ini.
4	Deklarasi fungsi post dengan parameter file.
5	Deklarasi variabel formData.
6	Menambahkan nilai variabel file ke formData dengan nama *key* file.
7	Memanggil *web service* dengan URI /TigaPilar/api/ticket/upload dengan HTTP *method* POST.
8	Deklarasi bahwa AngularJs tidak diperbolehkan mengubah data kita.
9	Mengeset nilai content-type dengan undefined.
12-13	Mengembalikan *response* jika berhasil.
14-15	Mengembalikan *response* jika gagal.

2. Implementasi *method* SendDataFactory.put(request) pada UploadService

Method SendDataFactory.put(request) digunakan untuk mengakses *web service* dengan URI /TigaPilar/api/ticket/send yang telah dibuat sebelumnya. Implementasi *method* ini dapat dilihat pada tabel 5.43, penjelasan *method* ini dapat dilihat pada tabel 5.44.

Tabel 5.43 Implementasi *method* SendDataFactory.put(request) pada UploadService

No	***Method* SendDataFactory.put(request) pada uploadService**
1	`angular.module('uploadService').factory('SendDataFactory',`
2	`  function($resource, RequestIdFactory) {`
3	`    return $resource(Context + '/api/ticket/send', null, {`
4	`      put: {`
5	`        method: 'PUT',`
6	`        params: {`
7	`          'requestId': RequestIdFactory.get()`
8	`        }`
9	`      }`
10	`    });`
11	`  });`

Tabel 5.44 Penjelasan *method* SendDatafactory.put(request) pada UploadService

No	Penjelasan
1	Deklarasi nama *module* uploadService, dan nama *factory* SendDataFactory.
2	Deklarasi pembuatan fungsi dengan parameter $resource dan RequestIdFactory.
3	Pengembalian nilai dari *factory* ini dengan mengakses URI /TigaPilar/api/ticket/send.
4	Deklarasi fungsi put.
5	Deklarasi HTTP *method* dengan nilai PUT.
6	Deklarasi parameter yang digunakan.
7	Deklarasi variabel requestId dengan nilai yang didapatkan dari pemanggilan *method* get() dari RequestIdFactory.

3. Implementasi *method* SaveTrialFactory.post(request) pada WeeklyService

Method `SaveTrialFactory.post(request)` digunakan untuk mengakses *web service* dengan URI /TigaPilar/api/trial yang telah dibuat sebelumnya. Implementasi *method* ini dapat dilihat pada tabel 5.45, penjelasan *method* ini dapat dilihat pada tabel 5.46.

Tabel 5.45 Implementasi *method* SaveTrialFactory.post(request) pada WeeklyService

No	***Method* SaveTrialFactory.post(request) pada weeklyService**
1	`angular.module('weeklyService').factory('SaveTrialFactory',`
2	`  function($resource, RequestIdFactory) {`
3	`    return $resource(Context+'/api/trial', null, {`
4	`      post: {`
5	`        method: 'POST',`
6	`        params: {`
7	`          'requestId': RequestIdFactory.get()`
8	`        }`
9	`      }`
10	`    });`
11	`  });`

Tabel 5.46 Penjelasan *method* SaveTrialfactory.post(request) pada WeeklyService

No	Penjelasan
1	Deklarasi nama *module* weeklyService, dan nama *factory* SaveTrialFactory.
2	Deklarasi pembuatan fungsi dengan parameter $resource dan RequestIdFactory.
3	Pengembalian nilai dari *factory* ini dengan mengakses URI /TigaPilar/api/trial.
4	Deklarasi fungsi post.
5	Deklarasi HTTP *method* dengan nilai POST.
6	Deklarasi parameter yang digunakan.

7	Deklarasi variabel requestId dengan nilai yang didapatkan dari pemanggilan *method* get() dari RequestIdFactory.

4. Implementasi *method* InputCostFactory.post(request) pada WeeklyService

Method `InputCostFactory.post(request)` digunakan untuk mengakses *web service* dengan URI /TigaPilar/api/trial/input-cost yang telah dibuat sebelumnya. Implementasi *method* ini dapat dilihat pada tabel 5.47, penjelasan *method* ini dapat dilihat pada tabel 5.48.

Tabel 5.47 Implementasi *method* InputCostFactory.post(request) pada WeeklyService

No	***Method* InputCostFactory.post(request) pada WeeklyService**
1	`angular.module('weeklyService').factory('InputCostFactory',`
2	`  function($resource, RequestIdFactory) {`
3	`    return $resource(Context+'/api/ticket/input-cost', null, {`
4	`      post: {`
5	`        method: 'POST',`
6	`        params: {`
7	`          'requestId': RequestIdFactory.get(),`
8	`          'ticketRegisterNumber':'@ticketRegisterNumber',`
9	`          'cost':'@cost'`
10	`        }`
11	`      }`
12	`    });`
13	`  });`

Tabel 5.48 Penjelasan *method* InputCostfactory.post(request) pada WeeklyService

No	Penjelasan
1	Deklarasi nama *module* weeklyService, dan nama *factory* InputCostFactory.
2	Deklarasi pembuatan fungsi dengan parameter $resource dan RequestIdFactory.
3	Pengembalian nilai dari *factory* ini dengan mengakses URI /TigaPilar/api/ticket/input-cost.
4	Deklarasi fungsi post.
5	Deklarasi HTTP *method* dengan nilai POST.
6	Deklarasi parameter yang digunakan.
7	Deklarasi variabel requestId dengan nilai yang didapatkan dari pemanggilan *method* get() dari RequestIdFactory.
8	Deklarasi variabel ticketRegisterNumber dengan nilai @ticketRegisterNumber
9	Deklarasi variabel cost dengan nilai @cost

5. Implementasi *method* InputPaymentFactory.put(request) pada DataService

Method `InputPaymentFactory.put(request)` digunakan untuk mengakses *web service* dengan URI /TigaPilar/api/ticket/{year}/{ticketRegisterNumber} yang

telah dibuat sebelumnya. Implementasi *method* ini dapat dilihat pada tabel 5.49, penjelasan *method* ini dapat dilihat pada tabel 5.50.

Tabel 5.49 Implementasi *method* InputPaymentFactory.put(request) pada DataService

No	***Method* InputPaymentFactory.put(request) pada DataService**

1
2
3
 	
4
5
6
7
8
9
10
11
	`angular.module('dataService').factory('InputPaymentFactory',`
	`  function($resource, RequestIdFactory) {`
	`  return $resource(Context +`
	`  '/api/ticket/:year/:ticketRegisterNumber',null, {`
	`  put: {`
	`  method: 'PUT',`
	`  params: {`
	`  'requestId': RequestIdFactory.get()`
	`  }`
	`  }`
	`  });`
	`  });`

Tabel 5.50 Penjelasan *method* InputPaymentfactory.put(request) pada DataService

No	Penjelasan
1	Deklarasi nama *module* dataService, dan nama *factory* InputPaymentFactory.
2	Deklarasi pembuatan fungsi dengan parameter $resource dan RequestIdFactory.
3	Pengembalian nilai dari *factory* ini dengan mengakses URI /TigaPilar/api/ticket/{year}/{ticketRegisterNumber}.
4	Deklarasi fungsi put.
5	Deklarasi HTTP *method* dengan nilai PUT.
6	Deklarasi parameter yang digunakan.
7	Deklarasi variabel requestId dengan nilai yang didapatkan dari pemanggilan *method* get() dari RequestIdFactory.

6. Implementasi *method* GetAvailableEvidenceFactory.get(request) pada EvidenceService

Method `GetAvailableEvidenceFactory.get(request)` diimplementasikan pada iterasi kedua yang digunakan untuk mengakses *web service* dengan URI /TigaPilar/api/ticket/available-evidence yang telah dibuat sebelumnya. Implementasi *method* ini dapat dilihat pada tabel 5.51, penjelasan *method* ini dapat dilihat pada tabel 5.52.

Tabel 5.51 Implementasi *method* GetAvailableEvidenceFactory.get(request) pada EvidenceService (iterasi kedua)

No	***Method* GetAvailableEvidenceFactory.get(request) pada EvidenceService**

1
 	
2
3
	`angular.module('evidenceService').factory('GetAvailableEvidenceFac`
	`tory',`
	`  function($resource, RequestIdFactory) {`

```
        return $resource(Context + '/api/ticket/available-
4       evidence', null, {
5         get: {
6           method: 'GET',
7           params: {
8             'requestId': RequestIdFactory.get(),
9             'year': '@year',
10            'month': '@month'
11          }
12        }
13      });
    });
```

Tabel 5.52 Penjelasan *method* GetAvailableEvidencefactory.get(request) pada EvidenceService (iterasi kedua)

No	Penjelasan
1	Deklarasi nama *module* evidenceService, dan nama *factory* GetAvailableEvidenceFactory.
2	Deklarasi pembuatan fungsi dengan parameter $resource dan RequestIdFactory.
3	Pengembalian nilai dari *factory* ini dengan mengakses URI /TigaPilar/api/ticket/available-evidence.
4	Deklarasi fungsi get.
5	Deklarasi HTTP *method* dengan nilai GET.
6	Deklarasi parameter yang digunakan.
7	Deklarasi variabel requestId dengan nilai yang didapatkan dari pemanggilan *method* get() dari RequestIdFactory.
8	Deklarasi variabel year dengan nilai @year
9	Deklarasi variabel month dengan nilai @month

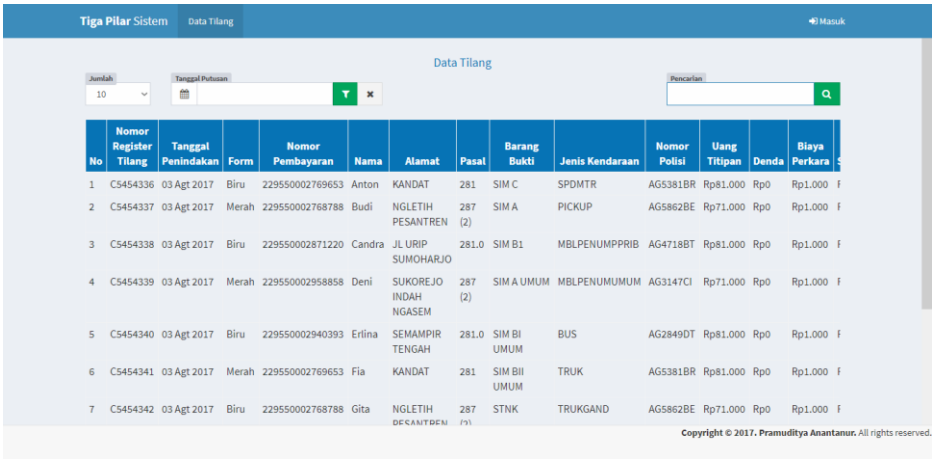
5.2.4 Implementasi Antarmuka

Implementasi antarmuka merupakan implementasi dari perancangan antarmuka sebelumnya. Implementasi antarmuka dibuat dari perancangan antarmuka pada iterasi pertama dan kedua.

- **Implementasi antarmuka iterasi pertama**

5.2.4.1 Implementasi antarmuka halaman *user* – data tilang

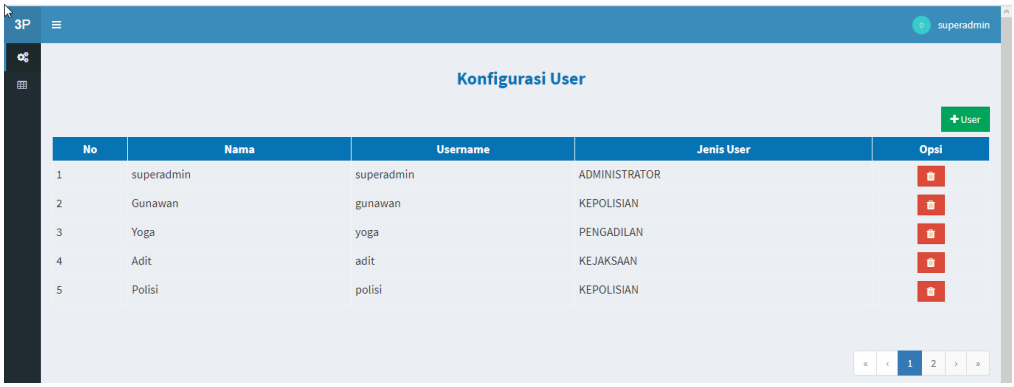
Antarmuka halaman *user* – data tilang merupakan antarmuka utama ketika sistem ini diakses, antarmuka ini digunakan untuk menampilkan data tilang ke masyarakat. Implementasi antarmuka halaman ini dapat dilihat pada gambar 5.43.



Gambar 5.43 Implementasi antarmuka halama *user* – data tilang

5.2.4.2 Implementasi antarmuka halaman *administrator* – konfigurasi

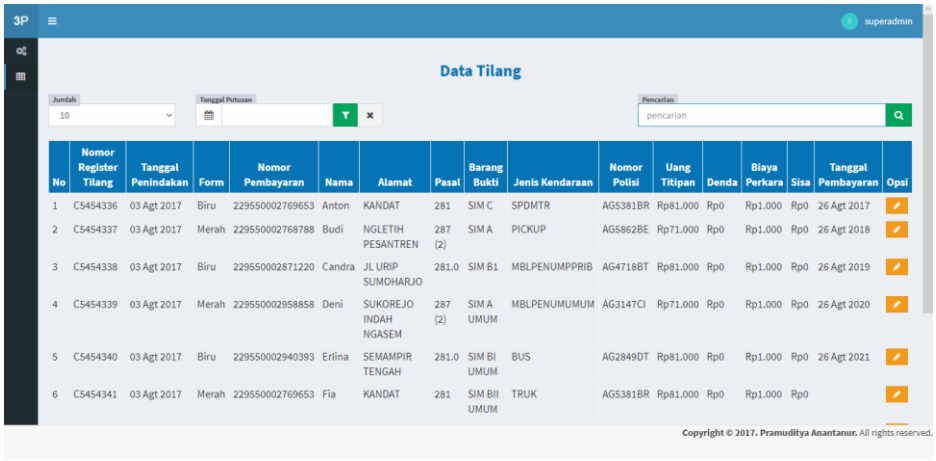
Antarmuka halaman *administrator* - konfigurasi merupakan antarmuka utama ketika aktor masuk sebagai *administrator*, antarmuka ini digunakan untuk menampilkan konfigurasi *user* yang diizinkan mengakses sistem. Implementasi antarmuka halaman ini dapat dilihat pada gambar 5.44.



Gambar 5.44 Implementasi antarmuka halaman *administrator* – konfigurasi

5.2.4.3 Implementasi antarmuka halaman *administrator* – data tilang

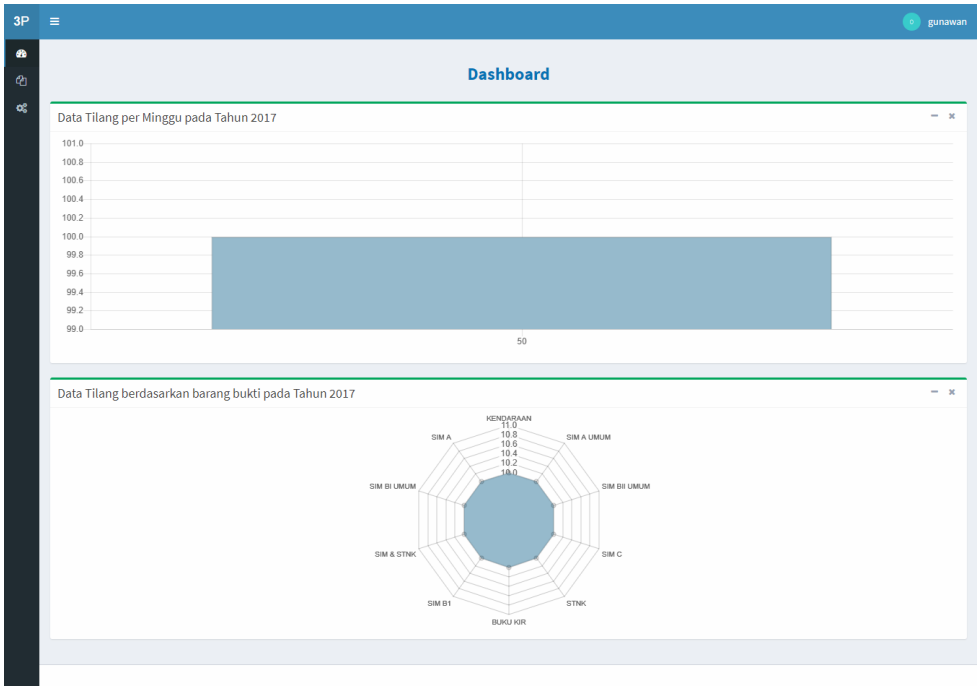
Antarmuka halaman *administrator* – data tilang merupakan antarmuka untuk menampilkan data tilang. Implementasi antarmuka halaman ini dapat dilihat pada gambar 5.45.



Gambar 5.45 Implementasi antarmuka halaman *administrator* – data tilang

5.2.4.4 Implementasi antarmuka halaman kepolisian – *dashboard*

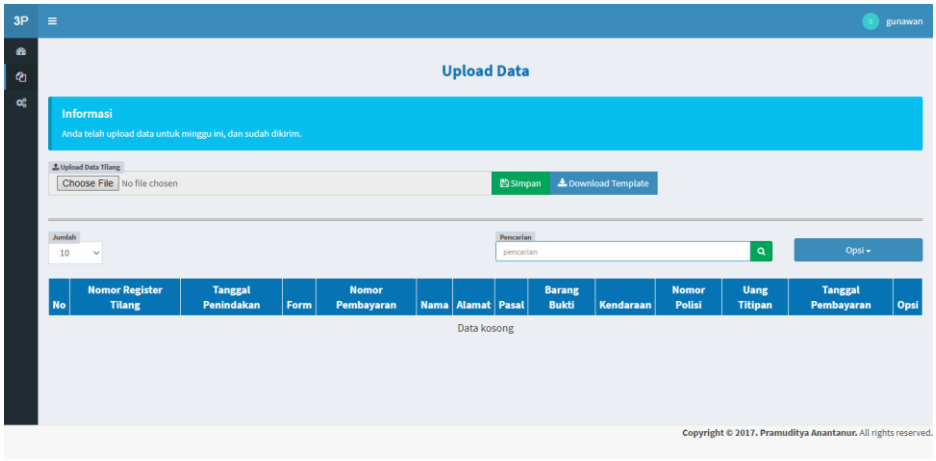
Antarmuka halaman kepolisian – *dashboard* merupakan antarmuka untuk menampilkan statistik data tilang berdasarkan minggu dan barang bukti. Implementasi antarmuka halaman ini dapat dilihat pada gambar 5.46.



Gambar 5.46 Implementasi antarmuka halaman kepolisian – *dashboard*

5.2.4.5 Implementasi antarmuka halaman kepolisian – *upload data*

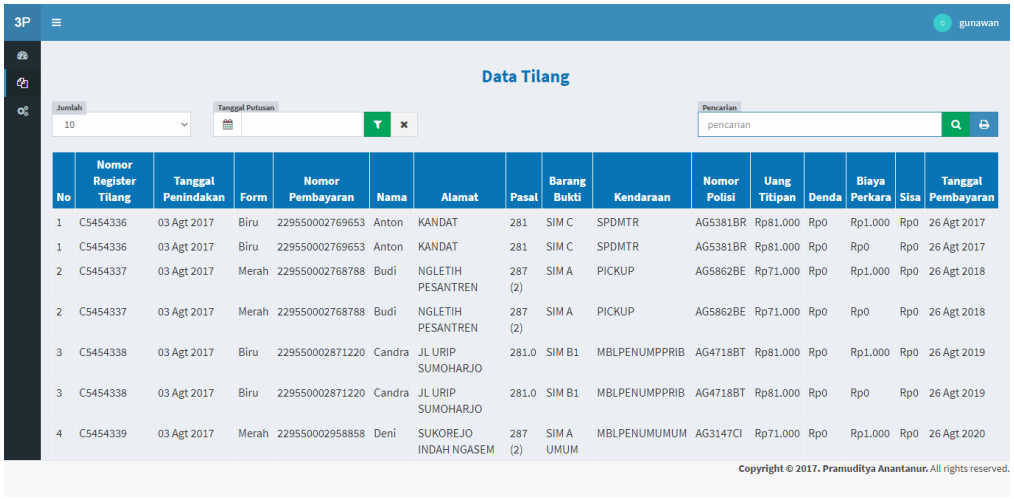
Antarmuka halaman kepolisian – *upload data* merupakan antarmuka untuk mengunggah *file* data tilang setiap minggunya. Implementasi antarmuka halaman ini dapat dilihat pada gambar 5.47.



Gambar 5.47 Implementasi antarmuka halaman kepolisian – *upload data*

5.2.4.6 Implementasi antarmuka halaman kepolisian – data tilang

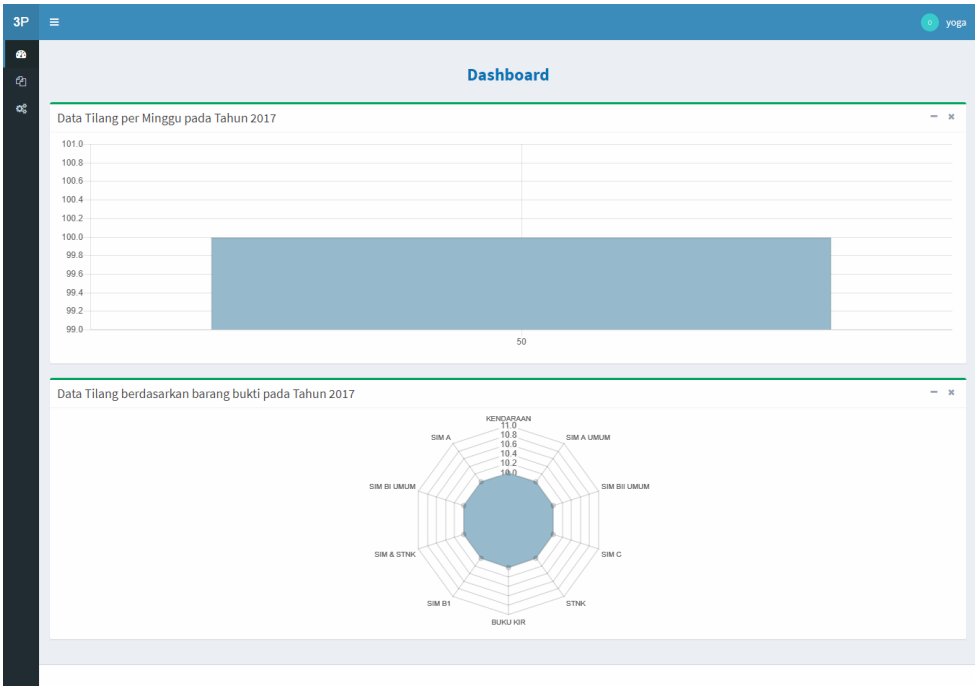
Antarmuka halaman kepolisian – data tilangmerupakan antarmuka untuk menampilkan data tilang. Implementasi antarmuka halaman ini dapat dilihat pada gambar 5.48.



Gambar 5.48 Implementasi antarmuka halaman kepolisian – data tilang

5.2.4.7 Implementasi antarmuka halaman pengadilan – *dashboard*

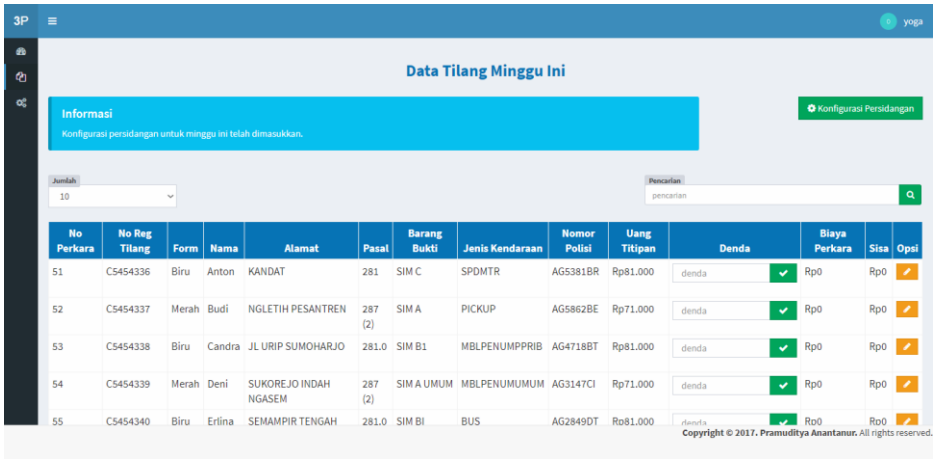
Antarmuka halaman pengadilan – *dashboard* merupakan antarmuka untuk menampilkan statistik data tilang dalam bentuk *chart* berdasarkan per minggunya dan barang bukti. Implementasi antarmuka halaman ini dapat dilihat pada gambar 5.49.



Gambar 5.49 Implementasi antarmuka halaman pengadilan – *dashboard*

5.2.4.8 Implementasi antarmuka halaman pengadilan – data minggu ini

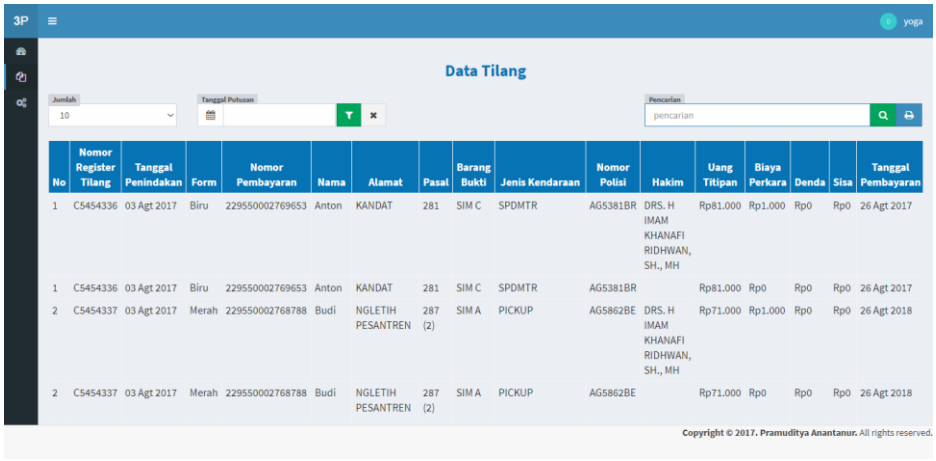
Antarmuka halaman pengadilan – data minggu ini merupakan antarmuka untuk menampilkan data tilang yang dikirimkan kepolisian dan belum diputus hakim. Implementasi antarmuka halaman ini dapat dilihat pada gambar 5.50.



Gambar 5.50 Implementasi antarmuka halaman pengadilan – data minggu ini

5.2.4.9 Implementasi antarmuka halaman pengadilan – data tilang

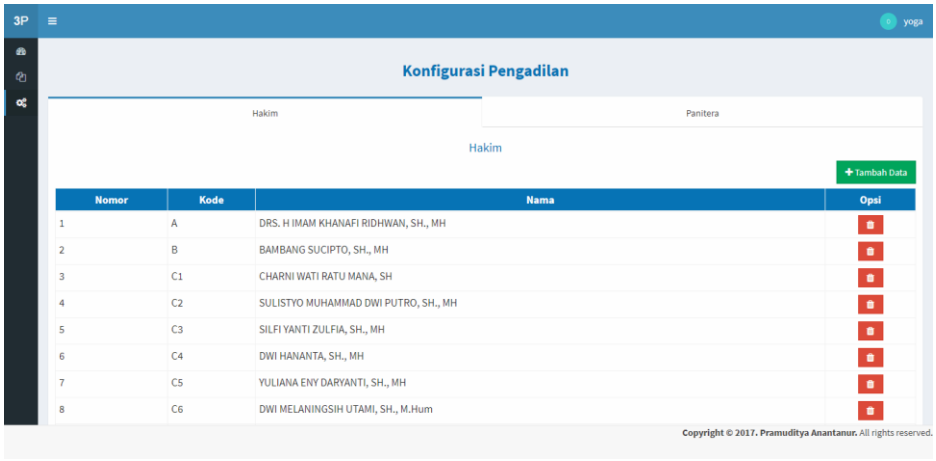
Antarmuka halaman pengadilan – data tilang merupakan antarmuka untuk menampilkan data tilang. Implementasi antarmuka halaman ini dapat dilihat pada gambar 5.51.



Gambar 5.51 Implementasi antarmuka halaman pengadilan – data tilang

5.2.4.10 Implementasi antarmuka halaman pengadilan – konfigurasi

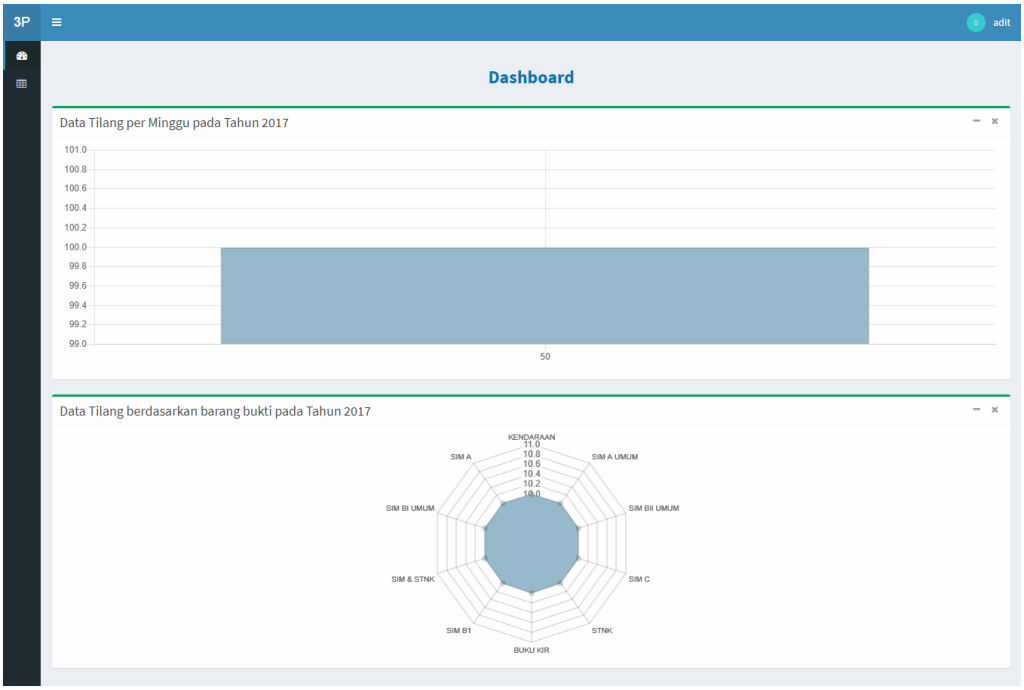
Antarmuka halaman pengadilan – konfigurasi merupakan antarmuka untuk melakukan konfigurasi hakim dan panitera yang ada pada pengadilan. Implementasi antarmuka halaman ini dapat dilihat pada gambar 5.52.



Gambar 5.52 Implementasi antarmuka halaman pengadilan – konfigurasi

5.2.4.11 Implementasi antarmuka halaman kejaksaan – *dashboard*

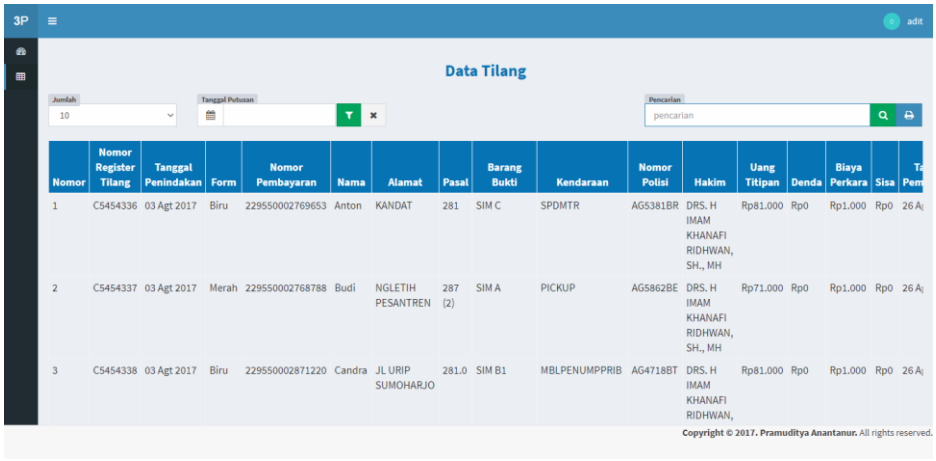
Antarmuka halaman kejaksaan – *dashboard* merupakan antarmuka untuk menampilkan statistik data tilang dalam bentuk *chart* berdasarkan per minggunya dan barang bukti. Implementasi antarmuka halaman ini dapat dilihat pada gambar 5.53.



Gambar 5.53 Implementasi antarmuka halaman kejaksaan – *dashboard*

5.2.4.12 Implementasi antarmuka halaman kejaksaan – data tilang

Antarmuka halaman kejaksaan – data tilang merupakan antarmuka untuk menampilkan data tilang dan melakukan konfirmasi pembayaran tilang. Implementasi antarmuka halaman ini dapat dilihat pada gambar 5.54.



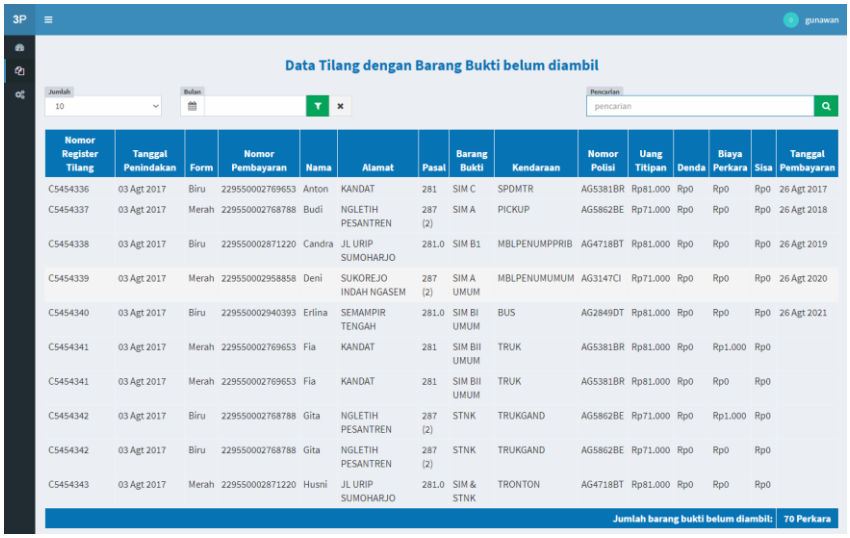
Gambar 5.54 Implementasi antarmuka halaman kejaksaan – data tilang

- **Implementasi antarmuka iterasi kedua**

Implementasi antarmuka pada iterasi kedua dilakukan karena terdapat penambahan fungsionalitas dari *user* yang mengharuskan terdapat penambahan antarmuka. Penambahan antarmuka antara lain dari operator kepolisian dan operator pengadilan. Operator kepolisian menginginkan ditampilkan data tilang dengan barang bukti yang belum diambil, sedangkan dari operator pengadilan menginginkan ditampilkan rekap data tilang dalam satu tahun.

5.2.4.13 Implementasi antarmuka halaman kepolisian – barang bukti (iterasi kedua)

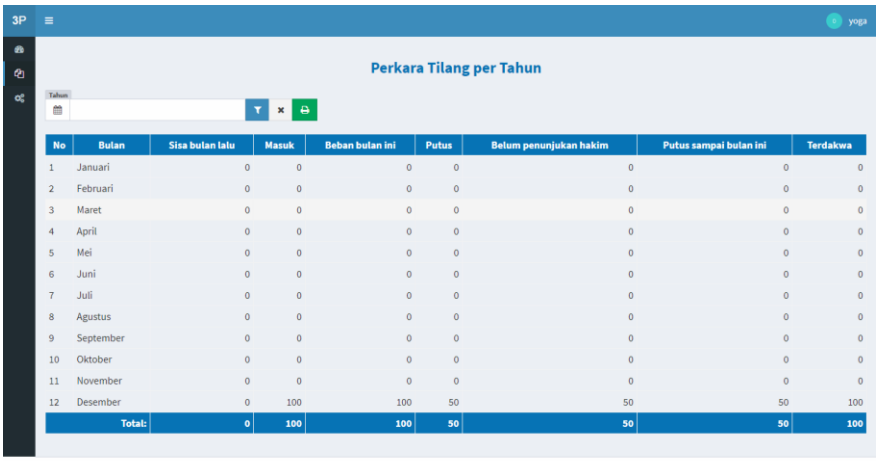
Antarmuka halaman kepolisian – barang bukti merupakan antarmuka untuk menampilkan data tilang dengan barang bukti yang belum diambil atau masih di kejaksaan. Implementasi antarmuka halaman ini dapat dilihat pada gambar 5.55.



Gambar 5.55 Implementasi antarmuka halaman kepolisian – barang bukti (iterasi kedua)

5.2.4.14 Implementasi antarmuka halaman pengadilan – perkara per tahun (iterasi kedua)

Antarmuka halaman pengadilan – perkara per tahun merupakan antarmuka untuk menampilkan rekap perkara per tahun. Implementasi antarmuka halaman ini dapat dilihat pada gambar 5.56.



Gambar 5.56 Implementasi antarmuka halaman pengadilan – perkara per tahun (iterasi kedua)